UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Lokalisation mittels Ultraschall

Bachelorarbeit
zur Erlangung des Grades
Bachelor of Science
im Studiengang Informatik

vorgelegt von

Marcus Opdenberg

Betreuer: Dr. Merten Joost, Institut für Physik, Fachbereich Naturwissenschaften, Universität Koblenz-Landau
Erstgutachter: Prof. Dr. Hannes Frey, Institut für Informatik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Mai 2016

Kurzfassung

Die folgende Arbeit zeigt eine Möglichkeit auf, Lokalisierung eines Objektes mittels Ultraschall zu realisieren. Dazu werden drei bis fünf im Raum verteilte Sensoren genutzt, um anhand von Distanzinformationen die Position eines Objekts relativ zu den Positionen der Sensoren zu bestimmen. Eine Besonderheit besteht dabei darin, dass die Sensoren nahezu beliebig in der Ebene[1] verteilt sein können. Ihre Anordnung wird vom System in der Kalibrierungsphase mit Unterstützung des Anwenders ermittelt. Dabei dürften ein gleichseitiges Dreieck, ein Quadrat oder Pentagramm je nach Sensoranzahl die besten Ergebnisse liefern. Um die relative Bewegung in eine Absolute zu übertragen, findet eine Umrechnung in Meter anhand der Taktung der Mikrocontroller, des Prescalers des verwendeten Timers und der Schallgeschwindigkeit statt.

[1] alle Sensoren müssen die gleiche Höhe haben

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet. Grafiken, die mit keiner Quelle gekennzeichnet sind, wurden selbst erstellt.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den 27. Januar 2016

Inhaltsverzeichnis

Kapitel 1

Bisherige Arbeiten von A.R. Jiménez and F. Seco

Im Folgenden wird kurz skizziert, welche Erkenntnisse und Fortschritte bezüglich Lokalisation bereits erlangt wurden. Im Sinne dieser Arbeit basieren die meisten der folgenden Projekte auf Ultraschall. Dabei lag der Fokus bislang hauptsächlich auf Automatisierungsaufgaben und intelligenten Gebäuden, denn auf einer realistischen Bewegung in der virtuellen Realität. Grundsätzlich lässt sich sagen, dass eine genaue Positionsbestimmung im Raum mittels Distanzen zu bekannten Punkten die Existenz von vier Messpunkten voraussetzt. Man bildet dazu Kugeln mit den Radien der Distanzen um die jeweiligen Punkte. Zwei Kugeln schneiden sich in einem Kreis, dieser wiederum geschnitten mit einer dritten Kugel liefert zwei Punkte. Diese Punkte ergeben, geschnitten mit einer vierten Kugel, den gesuchten Punkt. Dabei ist es notwendig, dass keine zwei Kugeln den gleichen Mittelpunkt haben und dass nicht mehr als zwei Kugelzentren auf je einer Geraden liegen. Bei einer Messung relativ zu drei Punkten, erhält man somit den Zielpunkt und dessen Spiegelung an der Ebene, die durch die besagten drei Punkte aufgespannt wird. In einigen Fällen kann dabei einer der Punkte durch Kenntniss der Umgebung ausgeschlossen werden. So reichen bei GPS drei Satelliten, um die Position auf der Erde zu bestimmen[1], da die Spiegelung des Punktes in Mitten des Weltalls liegen würde.

Die Autoren A.R. Jiménez and F. Seco vom Institut der industriellen Automatisierung in Madrid, berichten von den nun folgenden Projekten ihrer Arbeitsgruppe bzw. deren Vorgängern aus der Literatur. [Jim05]

[1] eine vierte Messung wird lediglich dazu benötigt, die Zeit zwischen den Satelliten und dem Empfänger zu synchronisieren

1.1 AT&T-Active Bat

Beim amerikanischen Telekommunikations-Anbieter AT&T wurde Anfang der 90er Jahre das "Active Bat" Sytem entwickelt. Dabei wurde das zu lokalisierende Objekt mit einem Ultraschall Emitter ausgestattet und die Umgebung mit Ultraschall Empfängern an der Decke versehen. Die Positionsberechnung erfolgte mittels Messung der Zeit zwischen Absenden und Empfang von Ultraschallwellen. Um die Zeit möglichst genau messen zu können, wurden Sender und Empfänger über Funk synchronisiert. Der Fehler lag bei 95% der Messungen unter 10cm.

1.2 Cricket

Das System "Cricket" nutzte nahezu die gleiche Vorgehensweise wie "Active Bat", nur dass die Rollen vertauscht wurden. So sendeten am Gebäude befestigte Transmitter die Ultraschallsignale aus, das Objekt empfing diese und berechnete darauf basierend die Position. Dies hatte vor allem Auswirkung auf die Privatsphäre, da das Gebäude nur die Position des Objektes kannte, sofern dies eben diese mitteilte.

1.3 Systeme ohne Ultraschall

Darüber hinaus gab es auch einige Projekte, die sich anderer Techniken als Ultraschall bedienten. So wurde der Vorgänger von "Active Bat" mittels Infrarot Leuchtfeuern realisiert, die sich an den Objekten befanden und Objekt-spezifische Licht-Impulse abgaben. Dabei konnte die Position dieser Objekte nur auf bestimmte Bereiche eingeschränkt werden.

Microsoft hingegen entwickelte das "RADAR" System, wobei das Objekt anhand der Signalstärke von Funkverkehr in WLAN Netzwerken geortet wurde. Dabei mussten die jeweiligen Signalstärken an mindestens zwei Basisstationen messbar sein. Dieses Projekt erreichte eine Genauigkeit von etwa vier Metern im zwei Dimensionalen, die Übertragung ins drei Dimensionale war aufgrund von Struktureigenschaften des Gebäudes äußerst problematisch.

Bei einem anderen Ansatz wurden Funkmodule an den Wänden verbaut, die in Zeitabständen Funksignale absetzten, die vom Objekt empfangen und mit aufmodulierter ID wieder abgesendet wurden. Somit konnten die Wand-Module die Laufzeit der Signale feststellen und daraus die Position berechnen.

1.4 Lokalisation für Rimho

Am Institut für industrielle Automatisierung in Madrid wurde ein System entwickelt, um den Lauf-Roboter "Rimho" zu lokalisieren und dessen Bewegungsablauf beim Laufen hinsichtlich der verbrauchten Energie zu optimieren. Dabei war sowohl die Position in der Ebene des Fußbodens als auch die aktuelle Höhe des Lauf-Roboters interessant. Um besagte Position zu bestimmen, wurden drei Ultraschall-Empfänger an einer Wand angebracht, während der Roboter einen omnidirektionalen Ultraschall-Sender erhielt. Auf Empfänger-Seite wurde dabei ein Bandpass-Filter um 40 kHz verwendet. Die Dauer des Signals wurde von der Mitte der jeweiligen Peaks aus gemessen, da diese die geringsten Unsicherheiten in der Zeit-Messung lieferten. Bei einer Aktualisierungs-Frequenz von etwa 40 Hz betrug der Fehler weniger als 1,5 mm bei einem Abstand von bis zu 6 m vor der Wand mit den Sensoren.

1.5 Lager-Automatisierung

Bei einem anderen Projekt wurde die Automatisierung eines Lagerhauses angestrebt. Dabei war es erforderlich, die Position eines Transportfahrzeugs bei einer Distanz von mehr als 20 m auf 1 bis 2 cm genau zu bestimmen. Zudem war es erforderlich, die Orientierung des Fahrzeugs mit einer Genauigkeit von etwa 1° zu erkennen. Dabei war es problematisch, das Schallsignal möglichst gleichmäßig in den Raum abzustrahlen. Ein Diffusor zur Streuung der Schallwellen kam nicht in Frage, da dieser das Signal stark dämpfen und somit die Reichweite einschränken würde. Daher wurden 2 zylindrische Anordnungen von jeweils 2 Ringen aus Ultraschall Emittern gewählt, die im Raum verteilt wurden und sequentiell sendeten.
Die Ultraschall-Empfänger des Transportfahrzeugs wurden ähnlich konstruiert, wobei hier ein Ring aus 8 Empfängern verwendet wurde. Durch die Ring-Anordnung war es einerseits möglich ein Signal aus jeder Richtung gleich gut zu empfangen, andererseits konnte, durch die Phasenverschiebung an den einzelnen Empfängern des Rings, gekoppelt mit der Kenntnis der Position des Fahrzeugs als auch den Positionen der Sender, auf die Orientierung des Fahrzeugs im Raum geschlossen werden. Aufgrund des sequentiellen Betriebs entstanden immer dann Fehler, wenn sich der Roboter bewegte, da die beiden notwendigen Distanzmessung von verschiedenen Orten aus erfolgten. Dieses Problem konnte durch Mittelwertbildung je zweier Messungen weitgehend beseitigt werden. Da die aktuelleste Position durch die Interpolation immer etwas veraltet ist, wird die tatsächliche, aktuelle Position als Verlängerung der letzten Messung angenommen. In Bereichen, in denen erhöhte Präzision gefordert wurde, konnte zudem die Geschwindigkeit gedrosselt werden, um einen geringeren Fehler zu erzielen.

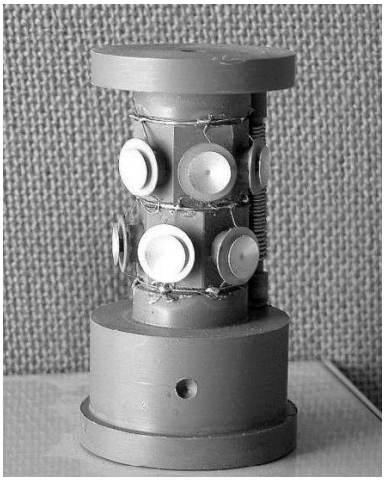

Abbildung 1.1: zylindrische Anordnung von Piezo Ultraschall-Sendern [Jim05]

1.6 Gewächshaus-Automatisierung

Im "Mino" genannten Projekt des Instituts für industrielle Automatisierung in Madrid, wurde die autonome Navigation von einem Düngemittel- und Pestizid Roboter angestrebt. Ein System, das den Roboter entlang einer Pflanzreihe auf Kurs hält und Kollisionen vermeiden kann, war bereits vorhanden, sodass die Entwicklung der Lokalisierung in einem Areal von über 50 × 50 m mit möglichst wenigen Sensoren im Vordergrund stand. Die Sender wurden in diesem Projekt ebenfalls an den Wänden befestigt und wiesen ein hemisphärisches Abstrahlmuster auf. Der Roboter erhielt 2 omnidirektionale Ultraschall-Empfänger, welche durch die Verbindung von je einem Ultraschall-Empfänger mit einem Kegel realisiert wurden.

Abbildung 1.2: omnidirektionaler Ultraschall-Empfänger [Jim05]

Bei einem Versuchsaufbau konnte der Roboter in einem $20 \times 20m$ großen Bereich mit Hilfe von 4 Sendern auf 5 cm genau lokalisiert werden.

1.7 Projekt Kaylite

Beim europäischen Projekt Kaylite ging es darum, das Äußere eines Schiffs oder andere Hüllstrukturen mittels eines Roboters in einer Art 3D Druck aus Verbundmaterialien zu fertigen. Dazu wurde eine maximale Abweichung der Austrittsdüse von 1mm gefordert, die jedoch durch Odometrie[2] des Roboters alleine nicht gewährleistet werden konnte. Dies wurde auf die Größe und die damit einhergehende Ungenauigkeit in Gelenken und Verbindungsstellen zurück geführt. Um die geforderte Präzision dennoch erreichen zu können, wurde die Austrittsdüse mittels Ultraschall lokalisiert und die entsprechende Position an die Kontrolleinheit des Roboters kommuniziert. Wie bereits beim Rimho Projekt, wurde auch hier das zu lokalisierende Objekt mit einem omnidirektionalen Ultraschall-Sender ausgestattet. Aufgrund der erwarteten Störungen durch Teile des Roboter-Arms oder des zu konstruierenden Gegenstandes, wurde eine redundante Anordnung von 8 Empfängern gewählt, anstelle der mindestens notwendigen dreien. Durch die Redundanz der Empfänger konnten zudem falsche, indirekte Messungen, solche die durch Reflexion an den beteiligten Komponenten entstehen, verworfen werden. Da der zu lokalisierende Roboterkopf konstruktionsbedingt immer innerhalb des durch die Sensoren gebildeten Quaders lag, reichten 3 Punkte, um die Position innerhalb des Quaders zu bestimmen.

Eine möglichst robuste Positionsbestimmung wurde durch die Verwendung von allen 56 möglichen Triplets aus den 8 Empfängern gebildet. Für jedes Triplet wurde die Position im Raum berechnet und diese in je eine sortierte Liste für X-,Y- und Z-Koordinate eingetragen. Von besagten Listen wurden jeweils die Ausreißer entfernt und die Mittelwerte in der Umgebung der Mediane gebildet, welche als robustes Ergebnis der Positionsbestimmung dienten. In einem 10 stündigen Test wichen die Messwerte nicht mehr als um -0,6 bis + 0,4 mm von der tatsächlichen Position ab.

[2] die eigene Messung / Schätzung seiner Bewegung

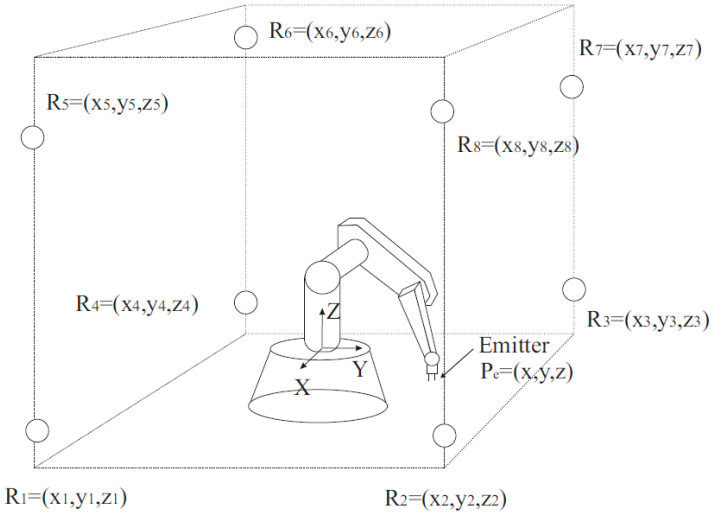

Abbildung 1.3: Skizze der Positionsbestimmung beim Projekt Kaylite [Jim05]

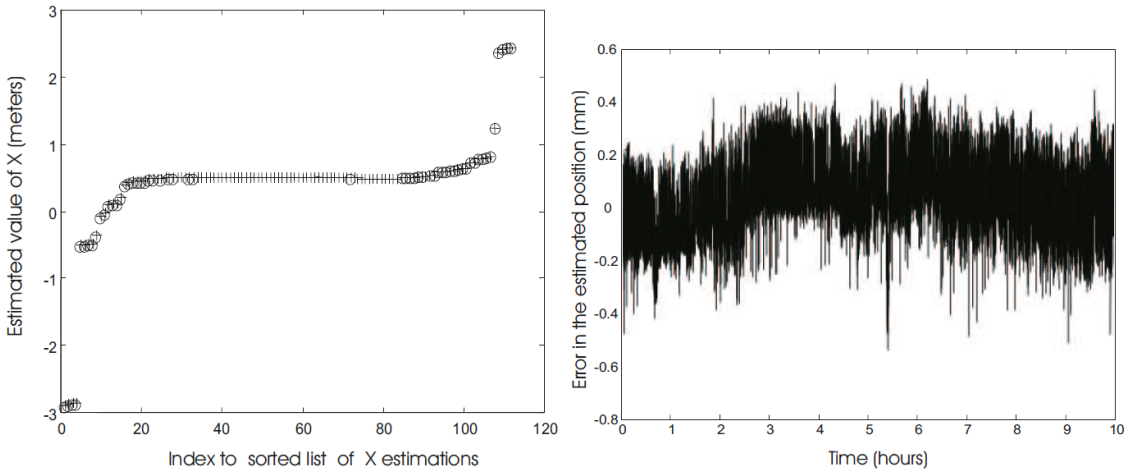

Abbildung 1.4: Messreihen beim Projekt Kaylite: links die Darstellung einer sortierten Liste der X-Koordinate einer Messung, rechts die euklidsche Abweichung der Messung von der tatsächlichen Position [Jim05]

Kapitel 2

Aufbau der Module

Ziel dieser Arbeit ist es, die Position eines Objektes im Raum mit Hilfe von Ultraschall zu ermitteln. Zu diesem Zweck müssen Module konstruiert werden, die:

1. Ultraschall senden
2. Ultraschall empfangen und die "Flugzeit" ermitteln
3. die Ultraschall-Module synchronisieren und die Verbindung zum Rechner herstellen

Die für diese Arbeit selbst konstruierten Module werden im Folgenden kursiv geschrieben, um diese von ihrer generellen Bedeutung unterscheiden zu können. Es gibt drei bis fünf *Sensoren*, die die Distanz zu einem *Transmitter* mittels Ultraschall ermitteln, sowie ein *Gateway*-Modul, welches die Schnittstelle zwischen Rechner, *Sensoren* und *Transmitter* realisiert. Das *Gateway*-Modul kommuniziert dabei mit Hilfe der seriellen Schnittstelle (RS232) mit dem Rechner bzw. mittels der SPI-Schnittstelle über Ethernet-Kabel mit den *Sensoren*, dem *Transmitter* und dem Funk-Modul. Es gibt zudem zwei *Transmitter*, die beide die SPI-Schnittstelle zur Kommunikation nutzen, dabei aber einmal drahtgebunden und einmal per Funk Daten empfangen. Der restliche Aufbau der *Transmitter* ist identisch.

Alle Module verfügen über einen mit 14 MHz getakteten Mikrocontroller der Art AtMega8 der Firma Atmel, deren Programmierung in der Sprache C mit Hilfe der Programmierumgebung AVRStudio 4.18 realisiert wurde. Die Spannungsversorgung erfolgt über USB-Akku-Packs[1] mit je 5V und 2,2 Ah.

Die *Sensoren* verfügen darüber hinaus über Ultraschall-Module des Typs HC-SR04. Das *Gateway*-Modul wird zudem durch einen Max232 Baustein erweitert,

[1] auch USB-Power-Bank genannt

um die Pegel der seriellen Schnittstelle zwischen Mikrocontroller und Rechner anzupassen. Das *Transmitter*-Modul nutzt außerdem eine integrierte H-Brückenschaltung des Typs L6202 - ein Bauteil, das sich mit TTL-Pegel mit bis zu 200 MHz schalten lässt und mit maximal 1,5A belastet werden kann.

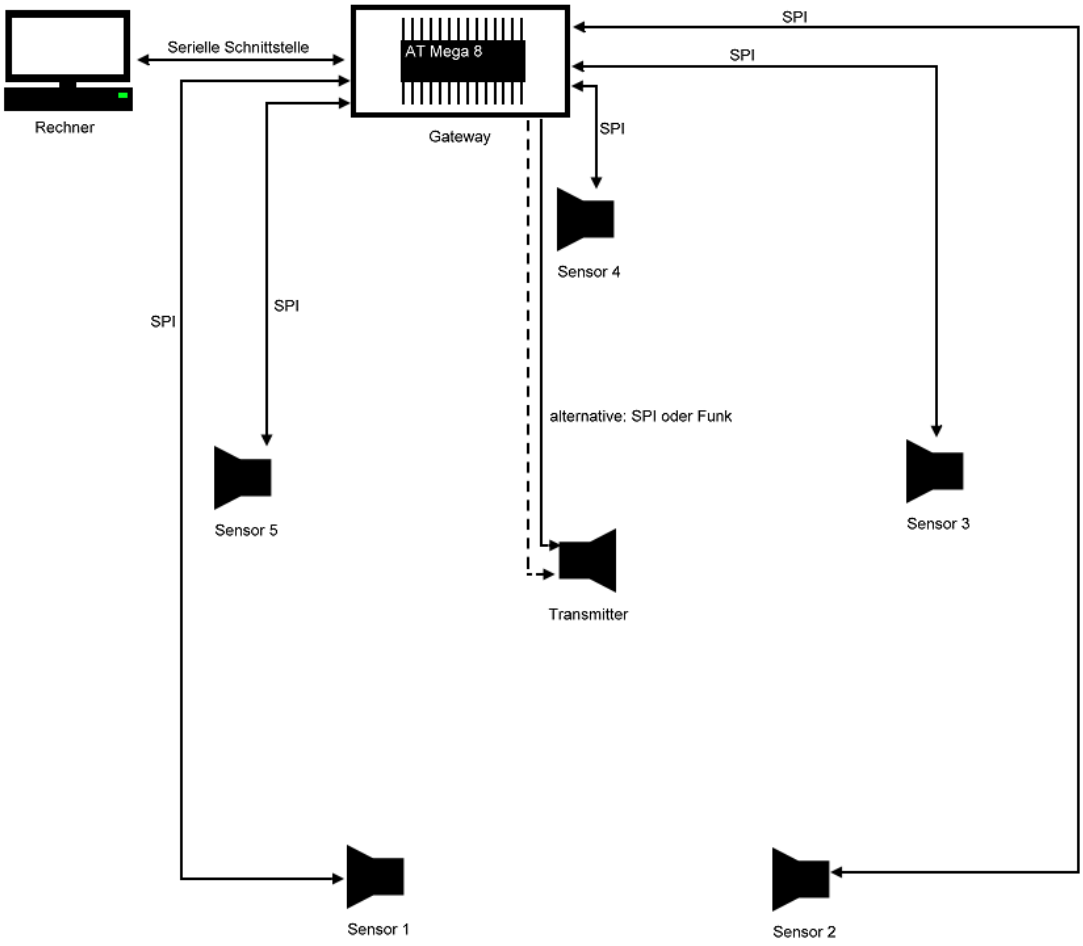

Abbildung 2.1: Überblick über die verwendeten Module (im Folgenden beschriebene Module sind teilweise in den Abgebildeten enthalten)

2.1 Kommunikation

Damit beide Module (jeweils *Sensor* und *Transmitter*) eine Distanz-Messung möglichst gleichzeitig starten können, wird der Zeitpunkt per SPI-Schnittstelle vom *Gateway*-Modul kommuniziert. Alternativ kann auch eine Funkverbindung vom *Gateway* zum *Transmitter* verwendet werden. Um der Verzögerung aufgrund des

Umwegs über die Funk-Module gerecht zu werden, findet die kabelgebundene Übertragung um etwa $800\mu s$ verzögert zur Funk-Übertragung statt. Der Ausgleich des zeitlichen Versatzes zwischen den verschiedenen *Transmittern* ist notwendig, um die beiden *Transmitter* im laufenden Betrieb austauschen zu können. Andernfalls würden die gemessenen Distanzen mit der Konfiguration des Koordinatensystems des jeweils anderen *Transmitters* kollidieren. Aufgrund der Störanfälligkeit einer Funkverbindung gegenüber einer kabelgebundenen Übertragung wird lediglich die einseitige Kommunikation von *Gateway*- zu *Transmitter*-Modul per Funk realisiert. Der Datenaustausch des *Gateways* mit den *Sensoren* erfolgt kabelgebunden über die SPI-Schnittstelle, da diese, einmal aufgestellt, ohnehin nicht mehr bewegt werden. Der *Transmitter*, den es zu lokalisieren gilt, soll dagegen möglichst vom Rest des Systems entkoppelt sein. Der Funkverbindung kommt dabei zu Gute, dass der *Transmitter*, im Gegensatz zu den *Sensoren*, nur eine eingehende Datenverbindung benötigt, wodurch Übertragungsfehler nur eine geringe Auswirkung haben. Der Synchronisierungsaufwand zwischen $Nachricht - gesendet$ und $Nachricht - empfangen$ entfällt somit. Die von den *Sensoren* ermittelten Distanz-Bytes werden im Folgenden einzeln über die SPI-Schnittstelle abgefragt und jeweils, gepaart mit der ID des Sensors, an den Rechner weitergeleitet (USART).

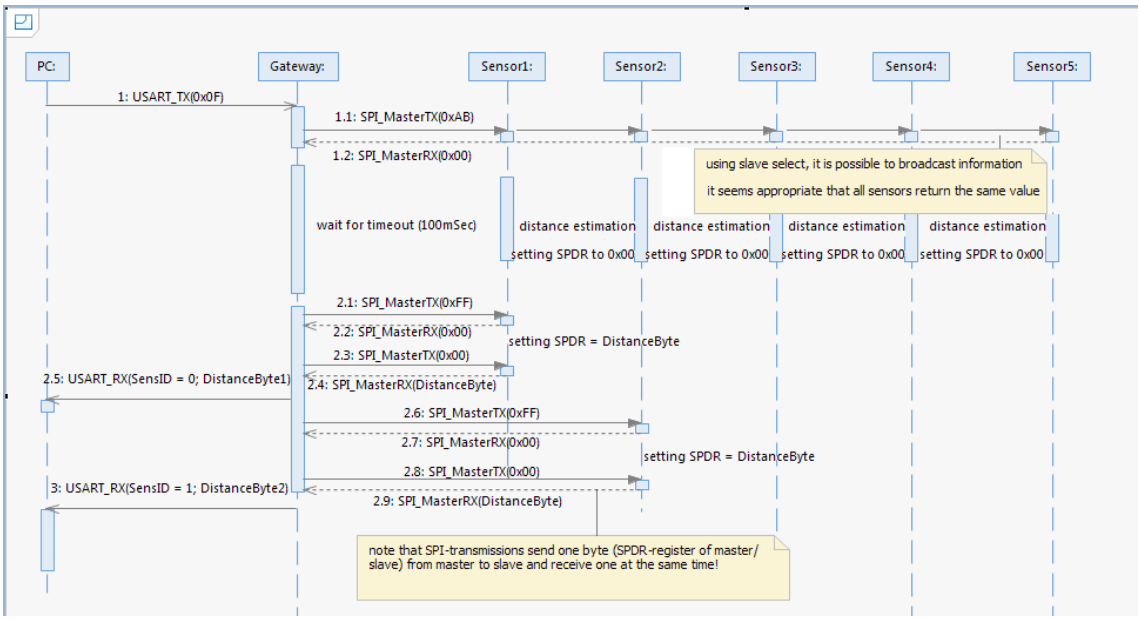

Abbildung 2.2: Sequenzdiagramm der Mikrocontroller-Kommunikation (größere Version im Anhang)

2.1.1 SPI-Schnittstelle

Das SPI-Interface nutzt vier Kanäle, die im Folgenden grob skizziert werden:

1. Clock (CLK) - das Takt-Signal wird vom SPI-Master generiert.

2. Master-in-Slave-out (MiSo) - das Byte aus dem SPDR-Register des Slaves wird hierüber empfangen (dabei können mehrere Slaves angesprochen werden, diese sollten dann aber das Gleiche senden, denn die einzelnen MiSo-Leitungen sind kurzgeschlossen).

3. Master-out-Slave-in (MoSi) - der Slave empfängt hierüber den Inhalt des SPDR-Registers des Masters.

4. Slave-Select ($\overline{SS}$) - Slave-Select ist low-active und signalisiert dem Slave, dass der Master eine Übertragung mit ihm intendiert. Dadurch ist es möglich mehrere Slaves an einem Master zu verwalten, was in dieser Arbeit auch verwendet wird. Dies hat unter anderem den Vorteil, dass allen *Sensoren* gleichzeitig der Start der Distanzmessung befohlen werden kann, während die Abfrage der gemessenen Distanzen einzelnd erfolgt. Dabei ist zu beachten, dass, sollte der Slave-Select-Kanal des Masters auf Masse gezogen werden, dieser in den Slave-Mode wechselt. Hier wurde der C-Port zur Realisierung der benötigten sechs (fünf *Sensoren*, ein *Transmitter*) Slave-Select-Leitungen verwendet. Das Funk-Modul verwendet eine externe Bibliothek, die jedoch ein SPI-Interface in Software realisiert. Da das Funk-Modul jedoch anders angesprochen werden muss[2], wurde dies an andere Pins, als die des Hardware-SPI-Interfaces angeschlossen.

2.1.2 Funk-Modul

Das Funk-Modul realisiert eine kabellose Verbindung zwischen den verwendeten Modulen mittels Frequency Shift Keying im 868 MHz Bereich. Da die Dokumentation teils unvollständig ist, wird zur Verwendung dieses Moduls auf eine Bibliothek des Nutzers "Benedikt" aus dem Forum "mikrokontroller.net" dankend zurück gegriffen.[K.07] Das Funk-Modul wird über die SPI-Schnittstelle angesprochen und erhält dabei Befehle von jeweils zwei Byte Länge, die für die Konfiguration, das Senden oder Empfangen von Daten verwendet werden.

Das *Gateway*-Modul lässt Nachrichten mit einer Nutzlast von zwei Byte verschicken. Diese Bytes entsprechen jeweils dem Befehl zum Starten einer Ultraschall-Messung ($0xAB$). Aus unbekannten Gründen wird bei besagten Funk-Modulen

[2] nicht nur ein Byte sondern immer zwei - z.B. um eine Konfiguration zu setzen oder ein Byte zu übertragen, dazu kommen Ein- und Aus-schalten des Transmitters respektive des Receivers

das letzte Byte einer Übertragung immer korrumpiert. Das erste Byte wird allerdings mit einer hohen Wahrscheinlichkeit richtig übertragen. Übertragungsfehler geschehen hauptsächlich aufgrund von räumlicher Nähe der Antennen mit anderen verwendeten Kabeln bzw. Geräten oder aufgrund von Verdeckung.

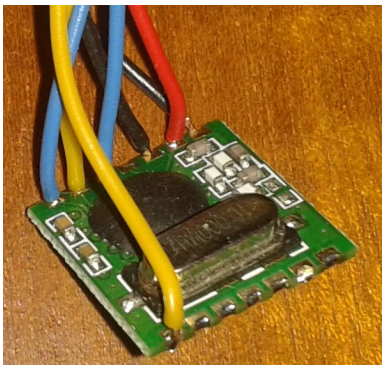

Abbildung 2.3: Funk-Modul RFM12B, Hope ltd.

2.2 Rechner

Beim Rechner wird eine serielle Schnittstelle vorausgesetzt. Sollte diese nicht vorhanden sein, so ließe sich diese mit Hilfe eines USB-Seriell-Wandler[3] nachrüsten. Die selbst geschriebene Software[4] wurde auf einem handelsüblichen Laptop von 2014 unter Windows 7 64 bit getestet. Der verwendete Rechner muss dabei 32 Bit-Anwendungen ausführen können. Bezüglich weiterer potentiell unterstützer Plattformen sei an dieser Stelle auf besagtes Software Kapitel verwiesen. Die Anforderungen hinsichtlich des Rechners sind vergleichsweise gering, höchst wahrscheinlich wäre ein Raspberry Pi bereits ausreichend, um die Positionsberechnung durchzuführen. Sollte eine VR-Brille zur Anwendung kommen, würde diese die Anforderungen vorgeben. Die Positionsberechnung dürfte dabei nicht merklich ins Gewicht fallen.

2.3 Ultraschall-Modul

Die hier verwendeten Ultraschall-Module des Typs HC-SR04 sind derart konzipiert, dass sie Ultraschall-Impulse senden und die Zeit bis zum Eintreffen derer Reflexionen messen. Die Reichweite beträgt dabei laut Hersteller maximal 4m. Die Variabilität von Geometrie und Material eines Objekts wirken sich auf dessen

[3] z.B. von Prolific
[4] beschrieben im Kapitel Software in C++

Reflexion aus und machen es somit äußerst schwierig, eindeutige Distanzen zu messen. Die vorgesehene Bewegung des Objekts macht derartige Messungen gänzlich unbrauchbar.

Als Konsequenz dessen werden Ultraschal-Messungen im Folgenden zwischen dem *Transmitter* und den *Sensoren* durchgeführt. Die Ultraschall-Emitter der *Sensoren* wurden entfernt, sodass diese die Messungen nicht beinflussen können. Ein *Sensor* misst schließlich die kürzeste Zeit, bis eine vermeintliche Reflexion eintrifft. Der *Transmitter* ist folglich das einzige Modul, das Ultraschall erzeugt.

An dieser Stelle soll zudem fest gehalten sein, dass die verwendeten Ultraschall-Module, obgleich ihrer gemeinsamen Bezeichnung, vermutlich aus mindestens drei verschiedenen Margen stammen und unterschiedlich reagieren. So erfolgt die unten genannte Ansteuerung bei allen identisch, die Module reagieren, zu mehreren per AND-Gate zusammenschaltet, jedoch unterschiedlich. Bei den Modulen mit vier Löchern und den Modulen mit zwei Löchern und unbeschriftetem Quarz, tritt am Rand des Empfangsbereich ein Effekt auf, der im folgenden als *Flattern* bezeichnet wird. Dabei schwankt die zeitliche Länge der Distanz-Antwort zwischen Null und einem Zeitwert, der einer realistischen Distanz entspricht. Die Fehlmessungen nehmen dabei zum Rand des Empfangsbereich hin deutlich zu und wirken sich somit am stärksten bei der anfänglichen Konfiguration des Koordinatensystems aus. Die Minderheit der Module, die chronologische zweite Marge, zeigt diesen Effekt nicht. Dafür schwanken die gemessenen Distanzen geringfügig mehr als die restlichen (um höchstens $\pm 2,5\%$).

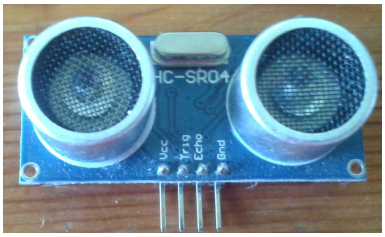

Abbildung 2.4: Ultraschall Modul, HC-SR04

2.3.1 Funktion/Aufgabe

Das Ultraschall-Modul startet bei einem anliegenden 10 ns High-Pegel am Trigger-Pin eine Ultraschall-Messung, indem 8 Ultraschall-Impulse gesendet werden und die Zeit bis zur Ankunft der (reflektierten) Impulse gemessen wird. Daraufhin wird ein zur Dauer proportionaler High-Pegel am Echo-Pin erzeugt. Wenn keine Reflektion erkannt wird, kann, je nach Art des Moduls, eine zufällige Distanz, keine Distanz oder ein unendliche Distanz ausgegeben werden. Letztere Ausgabe wird durch ein erneutes ansteuern des Moduls zurückgesetzt.

Vom Hersteller wird eine Reichweite von vier Metern angegeben, durch die verwendete, direkte Messung sollte die Reichweite jedoch auf maximal acht Meter ansteigen. Tests bei einer Distanz von etwa fünf Metern waren erfolgreich.

Es fiel auf, dass sich die Reichweite der Module bei einer zu geringen bzw. instabilen Versorgungsspannung auf etwa 60cm bei der Detektion von Reflexionen bzw. 120cm bei direkten Messungen beschränkt. Dazu mehr im Abschnitt über die *Sensoren*.

2.4 Gateway-Modul

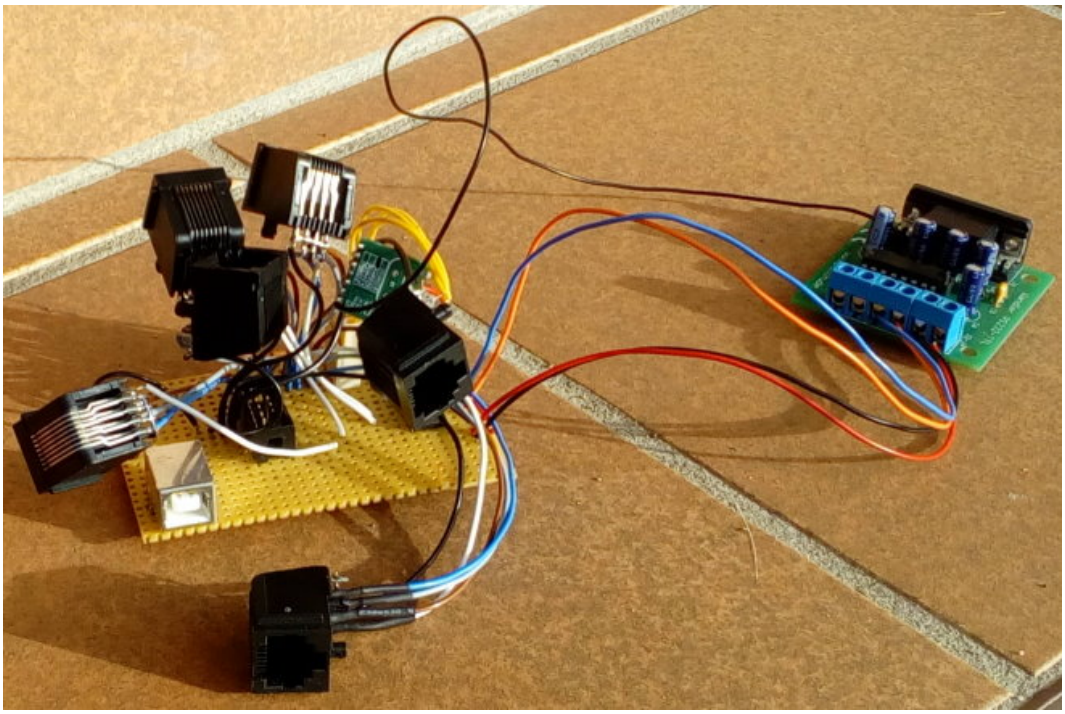

Abbildung 2.5: Bild des *Gateway*-Moduls

2.4.1 Schaltskizze

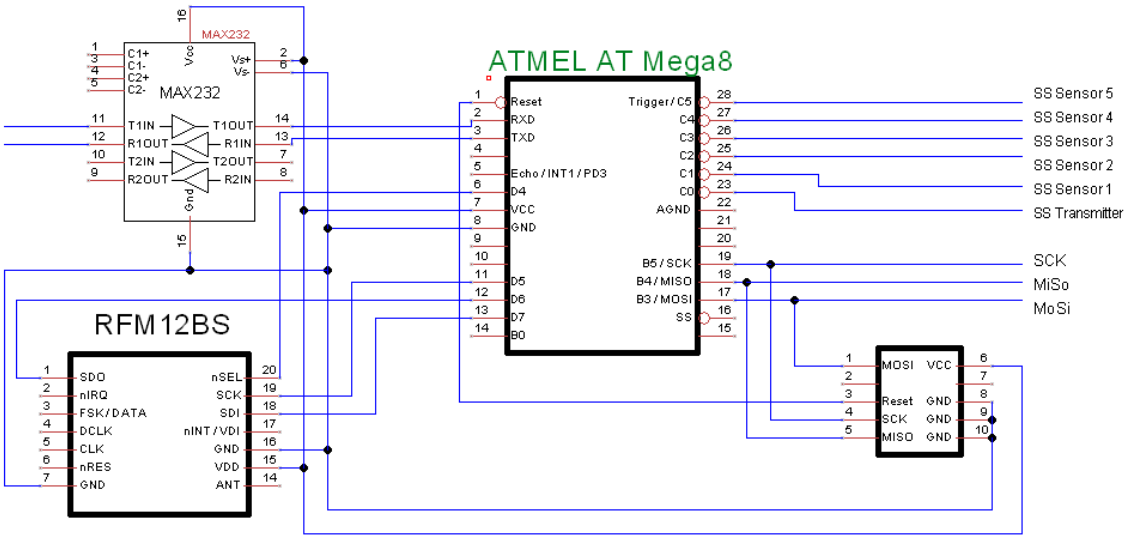

Abbildung 2.6: Schaltskizze des *Gateway*-Moduls

2.4.2 Funktion/Aufgabe

Das *Gateway*-Modul erhält über die serielle Schnittstelle vom Rechner einen Befehl zum Starten einer Distanzmessung in Form eines Bytes, bei dem die Bits der beteiligten Module gesetzt werden. Das niedrigste Bit addressiert den *Transmitter*, die fünf nächst höheren Bits addressieren die *Sensoren*, die zwei höchstwertigen Bits finden derzeit keine Verwendung und werden ignoriert. Die Slave-Select-Leitungen von *Transmitter* und *Sensoren* hängen am C-Port des *Gateway*-Mikrocontrollers, wobei ein Modul bei auf Masse gezogener Slave-Select-Leitung an der Kommunikation teilnimmt. Da alle Module an den gleichen Kanälen[5] hängen, wird das empfangene Byte invertiert[6] und in das Port-C-Register geschrieben, wodurch die gewünschten Kommunikationspartner direkt angesprochen werden. Der Befehl zum starten einer Messung wird somit an alle Module zeitgleich kommuniziert. Anschließend werden die gemessenen Distanzen der zuvor angesprochenen *Sensoren* jeweils nacheinander ausgelesen und per serielle Schnittstelle an den Rechner weitergeleitet.

[5] MiSo, MoSi und CLK
[6] mit $0xFF$ ver-x-odert

2.5 Transmitter-Modul

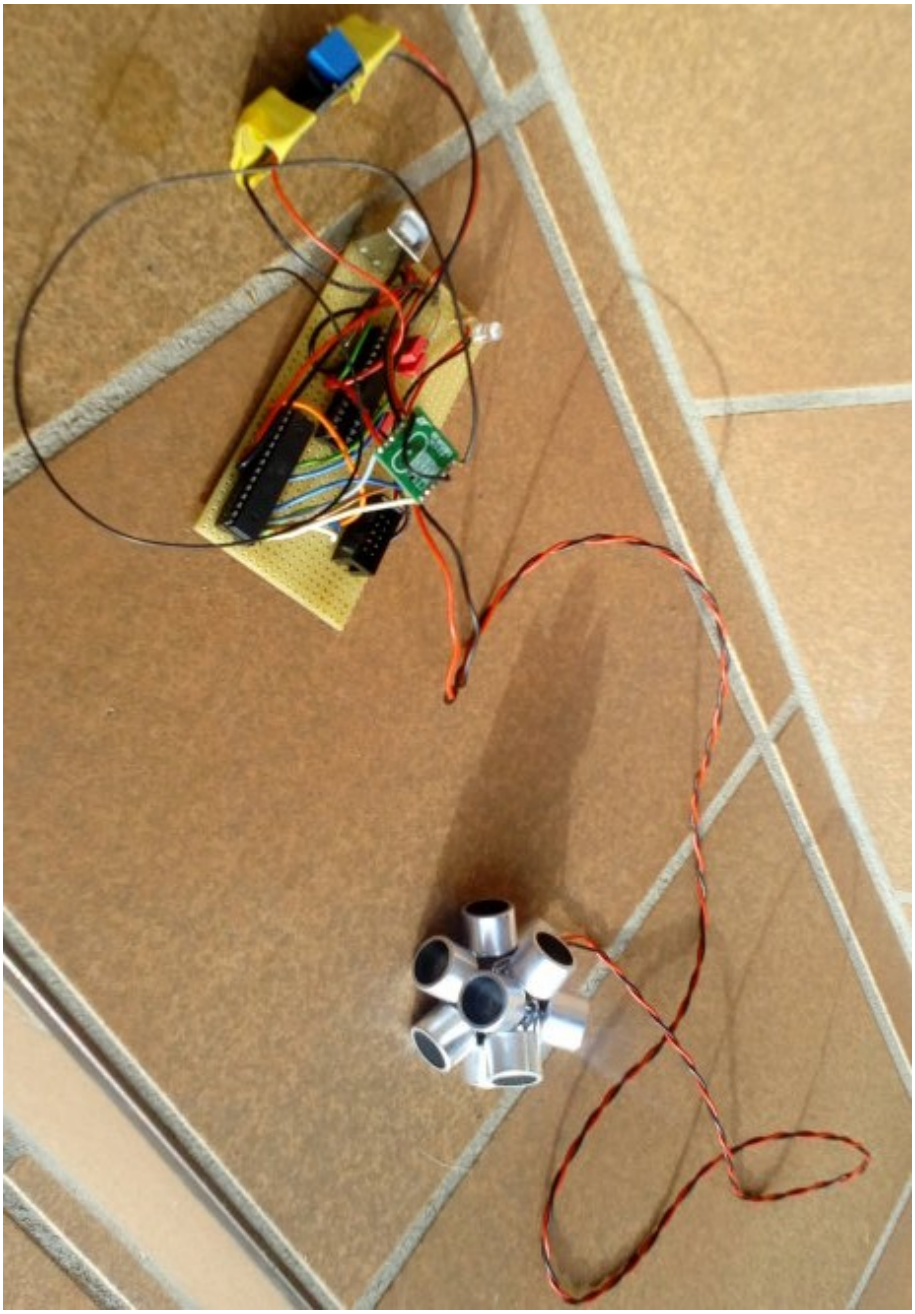

Abbildung 2.7: Bild des *Transmitter*-Moduls (hier: die Funk Version)

2.5.1 Schaltskizze

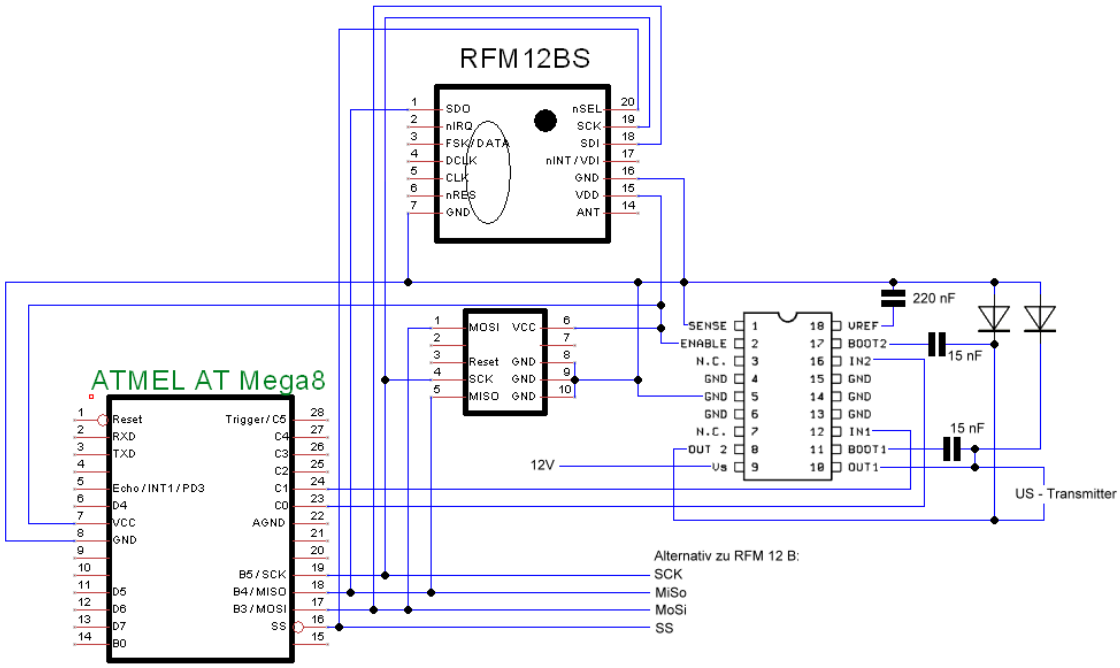

Abbildung 2.8: Schalt-Skizze des *Transmitter*-Moduls

2.5.2 Funktion/Aufgabe

Das *Transmitter*-Modul ist das Objekt, was mit dieser Arbeit lokalisiert werden soll. Es muss also Ultraschall möglichst gleichmäßig in alle Richtungen aussenden. Es wird jedoch die vereinfachende Annahme getroffen, dass der *Transmitter* zu jeder Zeit niedriger positioniert ist als die *Sensoren*. Dies hat die Vorteile, dass möglichst wenige Hindernisse in der Ausbreitungsrichtung existieren, die Ultraschall-Transmitter nur in einer Halbkugel nach oben angeordnet sein müssen und für die Berechnung der Position drei *Sensoren* ausreichen.

Das *Transmitter*-Modul erzeugt auf Befehl acht Ultraschall-Impulse der Frequenz 40 KHz. Zur Steigerung der Reichweite werden die Piezo-Ultraschall-Transmitter mit 12V, anstelle der ansonsten auf dem Modul verwendeten 5V angesteuert. Ursprünglich wurde die Spannung, inspiriert vom HC-SR04-Modul, aus dem die Transmitter ausgelötet wurden, von einem Max232 Baustein erzeugt. Jedoch brechen dessen Pegel bereits bei wenigen anzuregenden Ultraschall-Transmittern ein. Um gleichbleibende Pegel trotz zehn verwendeter Ultraschall-Transmitter zu gewährleisten, wurde auf eine zusätzliche Spannungsquelle und eine H-Brückenschaltung gesetzt. Die Spannungsquelle besteht aus einem Step-Up DC/DC-Wandler, der die

5V Eingangsspannung in einen Bereich einstellbar von 5 bis 35V (hier 12V) wandelt. Zur Erhöhung der Reichweite wäre es an dieser Stelle denkbar, die Ausgangsspannung des Step-Up Wandlers zu erhöhen, wobei fraglich ist, welche Spannung die Ultraschall-Transmitter verkraften können. Hinsichtlich der H-Brückenschaltung wurde aufgrund von Kompaktheit auf eine integrierte Version (L6202) zurückgegriffen. Da sich das Signal bei 40KHz bereits treppenartig aufbaute, wurden 15nF Kondensatoren an den Boot-Strap Eingängen des L6202 verbaut, wodurch das Rechteck-Signal des Mikrocontrollers reproduziert werden konnte. Die Beschaltung des L6202 orientert sich stark an der vorgeschlagenen Beschaltung des verwandten L6203 aus [Mis13].

2.6 Sensor-Modul

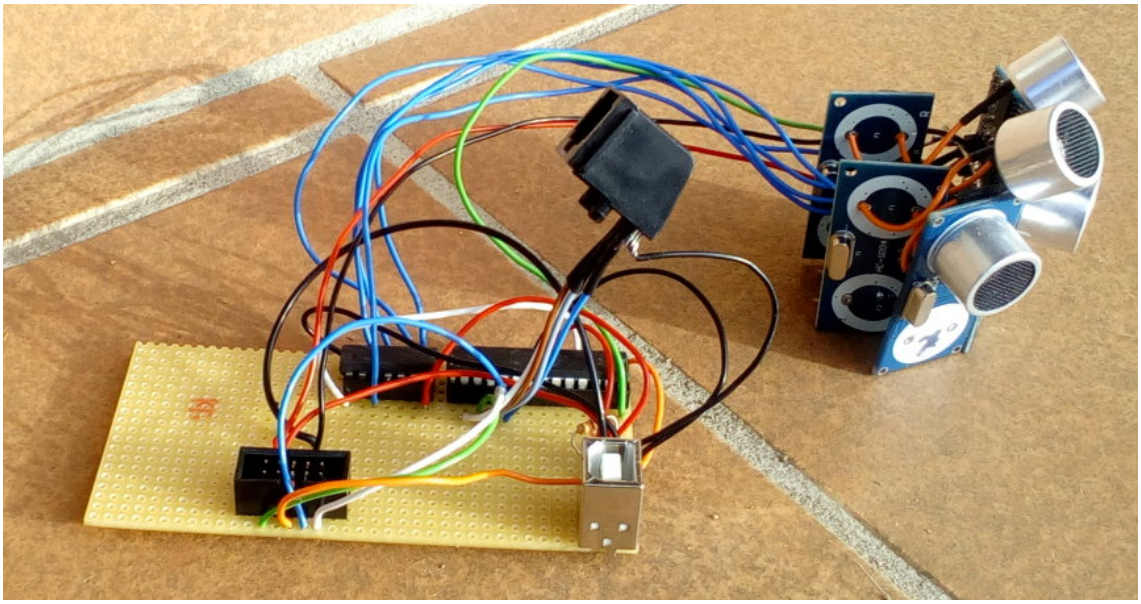

Abbildung 2.9: Bild des *Sensor*-Moduls (hier steht der HC-SR04-Cluster auf dem "Kopf")

2.6.1 Schaltskizze

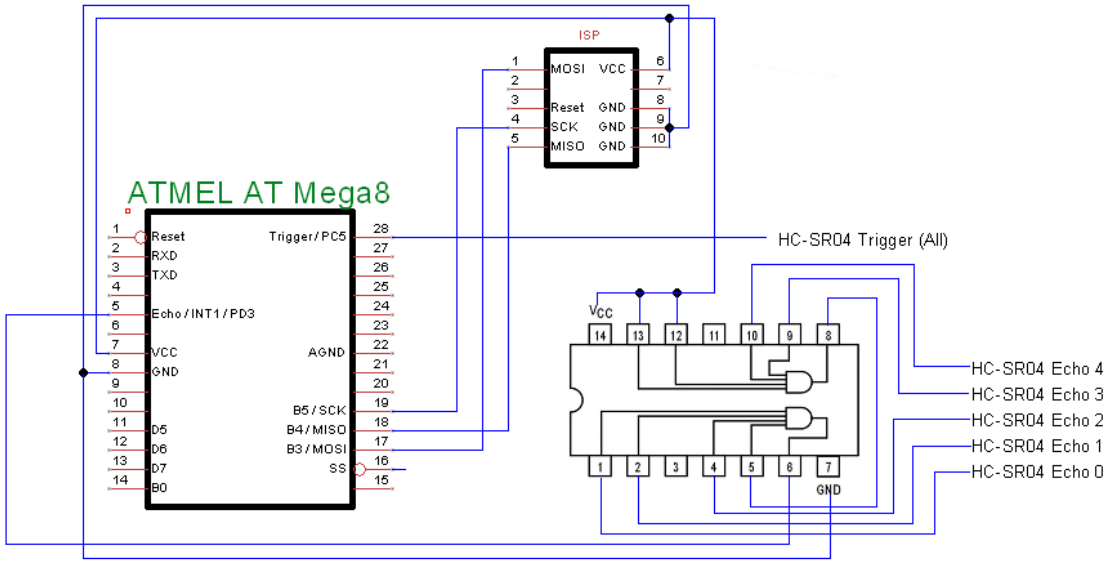

Abbildung 2.10: Schalt-Skizze des *Sensor*-Moduls

2.6.2 Funktion/Aufgabe

Die *Sensor*-Module und das *Transmitter*-Modul starten bei entsprechendem Befehl eine Ultraschall-Messung und liefern den gemessenen Wert auf Anfrage zurück. Der Distanz-Wert ergibt sich dadurch, dass der Mikrocontroller zunächst darauf wartet, dass das HC-SR04-Modul den Echo-Pin auf VCC[7] zieht. Danach wird der 16 Bit Timer (Timer 1) mit einem Prescaler von $\frac{1}{8}$ gestartet und wieder gestoppt, sobald der Echo-Pin auf Masse gezogen wird. Das High-Byte dieses Timers ist der Distanzwert, der im Folgenden verwendet wird. Das Low-Byte ist verrauscht und entfällt somit. Dieser Wert ist folglich keine Distanz in Metern, wird jedoch im späteren Verlauf von der rechnerseitigen Software in einen solchen umgewandelt[8]. An dieser Stelle sollte erwähnt werden, dass unterschiedliche Sensoren unterschiedliche Distanzen messen, deren Abweichungen aber durch konstante Faktoren angenähert werden können. Diese Faktoren werden im Rahmen der Kalibrierung ermittelt[9]. Vom High-Byte waren, bei Tests mit Distanzen unterhalb von 3,5 m, die niederwertigsten 7 Bit konstant für eine gegebene Distanz, das verbleibende Bit war bei der Maximal-Distanz null. Somit ergibt sich eine Präzision (Intervall-Größe) von $\frac{350}{128}$ cm ≈ 3 cm. Der Hersteller-Angabe folgend könnte eine maximale Distanz von

[7] Versorgungs-Spannung
[8] Anhand der Taktung des Controllers, des Timer-Prescalers und der Schallgeschwindigkeit
[9] siehe Abschnitt "Erstellung des Koordinatensystems" im Kapitel zur rechnerseitigen Software

8 Metern möglich sein[10]. Es wurden in räumlich bedingten Tests Distanzen bis etwa 6 m erfolgreich getestet. Dazu genaueres im Fazit.

Ein einzelnes Modul des Typs HC-SR04 hat bei einer direkten Distanzmessung[11] einen Öffnungswinkel von etwa 60°. Da in dieser Arbeit bis zu fünf *Sensoren* unterstützt werden sollten, ist bei gleichmäßiger Anordnung der *Sensoren* in einem Fünfeck ein Öffnungswinkel von etwa 110° pro *Sensor* notwendig. Dadurch, dass die meisten Tests in eher begrenzten Räumen statt fanden, konnte mit vier *Sensoren* der größte Bereich abgedeckt werden, sodass diese Konfiguration als die Wichtigste erschien. Daher wurden mehrere HC-SR04-Module zu einem *Sensor* Verbunden, wobei die Module in der Ebene in den Winkeln $0°, 45°$ und 90° (mit einer gewissen Fertigungs-Tolleranz) angeordnet sind. Bei dieser Verteilung ist der Öffnungswinkel auch groß genug für eine Konstellation mit fünf *Sensoren*. Dazwischen angeordnet befinden sich zwei weitere, unter 45° nach unten gerichtete HC-SR04-Module, wobei der von diesen Teil-Modulen abgedeckte Bereich vergleichsweise unbedeutend erscheint. Entsprechend groß sind hierbei die Fertigungs-Tolleranzen. Die HC-SR04-Module sind dabei über den VCC-, GND- und Trigger-Pin zu einem Konstrukt verbunden, die Echo-Pins werden jeweils zu einem 2x4-Fach AND-Gate der Art IC7421 geführt und darüber mit dem Microcontroller verbunden. Dies erspart dem Controller einige Arbeit hinsichtlich der Auswertung, erzeugt jedoch das bereits beschriebene "Flattern". Eine denkbare Alternative wäre es, die Echo-Pins direkt mit dem Microcontroller zu verbinden und die Auswertung, welches Modul den jeweiligen Echo-Pin zu erst wieder auf Masse zieht, per Software zu ermitteln. Dabei könnte dann auch grob die Richtung bestimmt werden. Dadurch ließ sich in Tests das Flattern der modifizierten *Sensoren* beseitigen, wobei jedoch deren Präzision litt, sodass diese Konstruktion wieder verworfen wurde.

Ein weiteres Problem ist die im Abschnitt des HC-SR04-Moduls bereits angesprochene Problematik der Stromversorgung dieser Module. Die verwendeten 5V Akku-Packs besitzen einen Lithium-Akku mit $3,7V$ und erzeugen die für USB benötigten $5V$ mittels eines Step-Up-Wandlers. Dieser benötigt jedoch eine Grundlast, um die Spannung sauber zu regeln. Anfangs schien eine $9cd$ Power-LED mit einem Strom von $20mA$ auszureichen. Da das gleiche Problem später jedoch wieder auftrat, wurde ein 100Ω Widerstand direkt zwischen VCC und GND geschaltet, was die Grundlast auf $20mA + 50mA$ anhebt. Drei der Akku-Packs eignen sich trotz dieser Maßnahme nicht zum Betrieb an den *Sensoren* und werden daher für das *Gateway* und die *Transmitter* verwendet.

[10] Angabe beläuft sich auch 4 m für Reflexionen
[11] bei Reflexion wesentlich weniger - etwa 15°

Kapitel 3

Software in C++

Um die rechnerseitige Anwendung (im Folgenden *Software* genannt), insbesondere die grafische Benutzeroberfläche (GUI), auf dem Rechner zu realisieren, wird auf das Qt-Framework zurück gegriffen. Dies stellt neben etlichen Code-Beispielen, auf die an verschiedenen Stellen dankend zurückgegriffen wird, auch ein möglichst hohes Maß an Portabilität[1] in Aussicht. Denkbar wäre auch eine Portierung auf einen Raspberry Pi, wobei ein solcher nicht die Kapazitäten hätte, eine VR-Brille mit den notwendigen Bild-Daten zu versorgen. Für weniger komplexe Aufgaben wäre der Ansatz womöglich interessant.

Die gesamte *Software* basiert in ihrem Kern auf der Qt-Beispiel-Applikation "Blocking Master'[Shi12]'. In dieser Anwendung wird beschrieben, wie mit Hilfe von Threads eine GUI parallel mit blockierenden, wiederholten Abrufen der seriellen Schnittstelle betrieben werden kann. Bei ersten Versuchen in einem einzigen Thread, fiel eine "einfrierende" GUI bei wiederholender Abfrage der seriellen Schnittstelle (polling) auf. Dies liegt daran, dass für die GUI regelmäßig ein sogenannter "Event-Loop" durchlaufen werden muss, um diese zu aktualisieren, was jedoch nur mit einem parallelen Ansatz realisierbar ist.

Des Weiteren wurde auf Teile der Qt-Beispiel-Applikation "Basic Drawing"[Plc13] zurückgegriffen, um die grafische Ausgabe der *Sensor*-Positionen und des detektierten *Objekts*[2] zu realisieren.

[1] die *Software* wurde unter Windows 7 entwickelt, sollte aber auch für Linux kompiliert werden können, ohne den Quellcode anpassen zu müssen.

[2] je ein Punkt pro Triplet

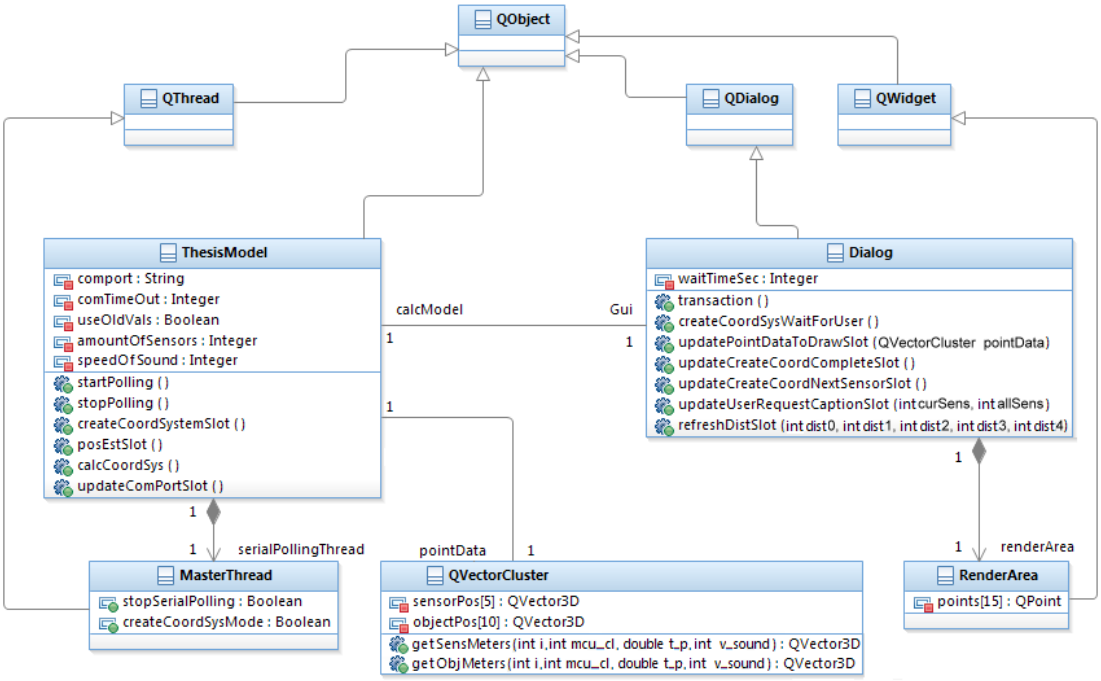

Abbildung 3.1: Klassendiagramm der rechnerseitigen *Software* in C++. Alle abgebildeten Attribute verfügen über getter und setter (nicht notwendigerweise Beides). Für das Gesamt-Verständis weniger bedeutende Attribute wurden weggelassen.

Die *Software* setzt sich primär aus der *GUI*-Klasse (*Dialog*) und der *Modell*-Klasse (*ThesisModel*) zusammen, zwischen denen bidirektional navigiert werden kann. Aufgrund der gegenseitigen Abhängigkeit muss auf Zeiger zurück gegriffen werden, um die Navigation zu realisieren. Da die Zeiger bei der Initialisierung noch nicht gesetzt sind, können nicht alle "connect"-Aufrufe[3] in den jeweiligen Konstruktoren erfolgen und werden somit in die entsprechenden "setter" für die Zeiger verlagert.

3.1 GUI-Klasse

Eine GUI arbeitet im Grundsatz einen Event-Loop ab. Dies bedeutet dass periodisch überprüft wird, welche Signale emittiert wurden[4] und welchen Slots diese zugeordnet sind.

[3] Erklärung im Abschnitt "Slots"

[4] geklickte Buttons, von Methoden generierte Signale, Timer oder Ähnliches

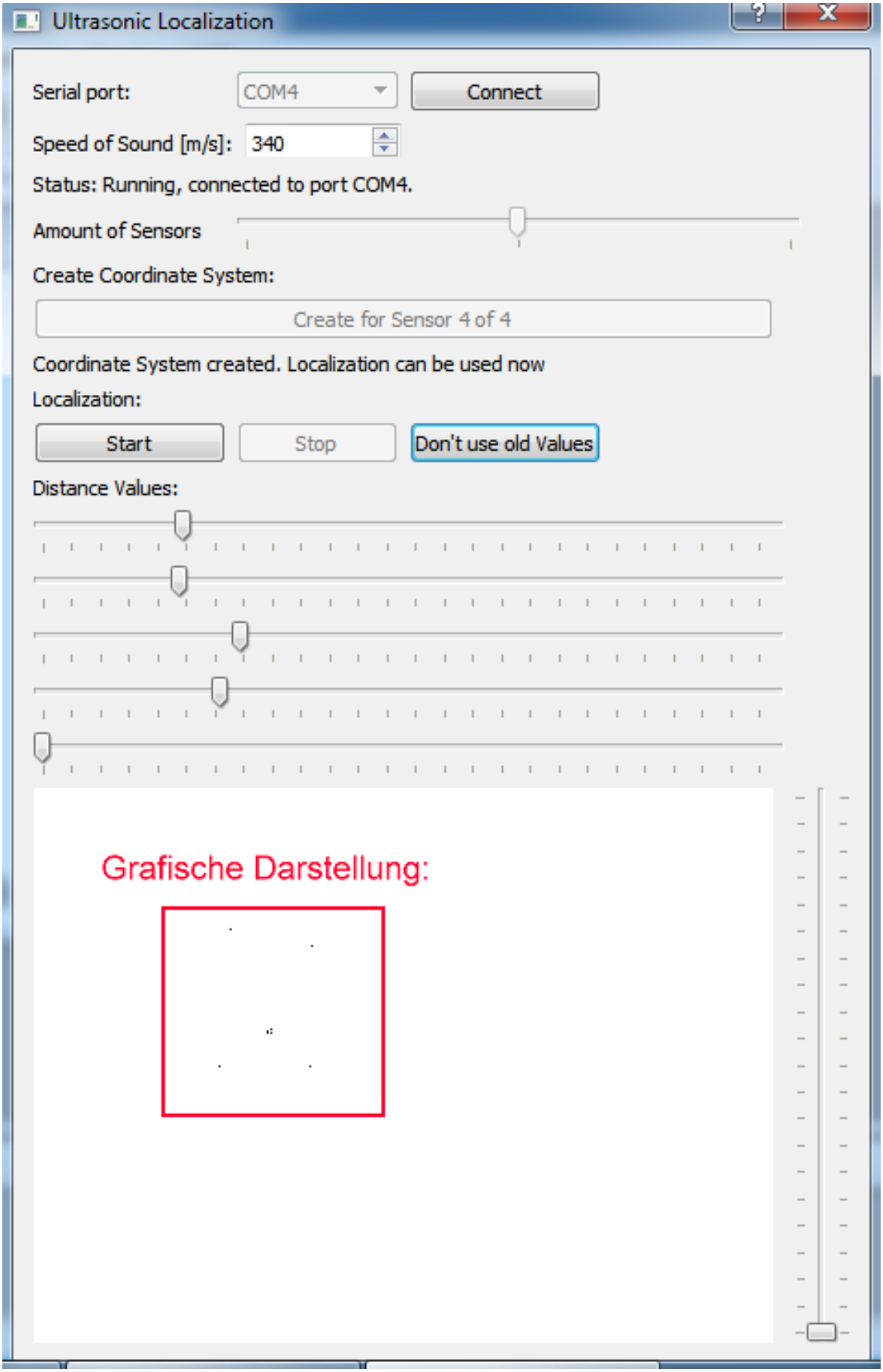

Abbildung 3.2: Screenshot der GUI mit markierter grafischen Ausgabe der Punkte

3.1.1 Überblick über die GUI

1. Das Drop-Down-Menü neben dem Connect-Button listet alle verfügbaren seriellen Schnittstellen auf.

2. Die Betätigung des Connect-Buttons startet die Kommunikation mit dem *Gateway*-Modul über die serielle Schnittstelle und führt eine Messung durch. Dabei wird die serielle Schnittstelle mit einem standard Timeout von $1000\mu s$ konfiguriert. Sollte die Verbindung nicht gelingen, so wird dies durch das Status-Label angezeigt.

3. Die Schallgeschwindigkeit wird verwendet, um einen Skalierungsfaktor zu bestimmen, der die gemessenen, zu den Distanzen korrespondierenden, Zeitwerte in Distanzen in Metern umrechnet. Über diesen Parameter lässt sich das System den äußeren Umständen[5] anpassen.

4. Der Start-Button startet Distanz-Messungen. Sollte das Koordinatensystem noch nicht konfiguriert sein, so erhält man lediglich die Ausgabe der einzelnen *Sensoren* auf den Distanz-Schiebereglern. Dies ist vor allem für einen initialen Test der *Sensoren* interessant. Nach erfolgreicher Kalibrierung wird die Positionsmessung durch diesen Button gestartet, wobei die ermittelte Position in der X-Y-Ebene in der unten befindlichen Grafik und die Z-Position anhand des rechts daneben liegenden Schiebereglers dagestellt wird. Die grafische Anzeige gibt dabei die Position der *Sensoren* und die Positionen der einzelnen Mess-Triplets[6] aus. Die Z-Koordinate wird lediglich durch das Ergebnis des ersten Triplets gesteuert.

5. Der Create-Coordinate-System-Button startet den Konfigurations-Vorgang oder setzt diesen für den nächsten *Sensor* fort. Nach Betätigung verbleiben 10 Sekunden, um den *Transmitter* zum betreffenden *Sensor* zu transportieren. Der *Transmitter* muss dort verbleiben, bis die Messreihe abgeschlossen ist und darf lediglich rotiert werden. Die Rotation dient dazu, alle *Sensoren* zu erreichen, da dies häufig nicht aus jeder Lage des *Transmitters* für alle *Sensoren* gelingt.

6. Der Don't-use-old-Values-Button schaltet zwischen zwei Modi um. Ist der alte-Werte-Modus inaktiv, so bedeutet dies, dass nur aktuelle Messungen in der Grafik dagestellt werden. Ist eine der Distanzen eines Triplets Null, so wird dessen Ergebnis nicht dagestellt. Ist besagter Modus aktiv, so wird eine Null-Messung durch den letzten gemessenen Wert größer Null zwischen dem

[5] Abweichungen der Sensoren, Temperatur, Luftfeuchtigkeit
[6] siehe Positionsberechnung

Transmitter und dem betreffenden *Sensor* ersetzt. Wenn viele Messungen nacheinander ausfallen, können die Ergebnisse der Triplets dadurch stark auseinander driften.

7. Die Distanz-Schieberegler dienen Primär der Fehler-Erkennung und -Behebung. Der oberste Regler repräsentiert den ersten *Sensor*, dieser wiederrum ist der *Sensor* dessen RJ45-Stecker am nächsten zum Mikrocontroller des *Gateway*-Moduls liegt.

3.1.2 Signale

Signale haben eine intuitive Funktion. Sie zeigen an, dass ein bestimmes Ereignis stattgefunden hat. Neben Signalen bestehender Klassen, lassen sich auch eigene Signale im Header im Bereich "signals:" definieren. Dabei besteht die Möglichkeit, Parameter für den auszuführenden Slot zu übergeben. Arrays werden dabei von Qt jedoch nur bedingt unterstützt, da spezifische Datenstrukturen bevorzugt werden. Soll eine Klasse auf Qt-Funktionen wie Signal oder Slot zurück greifen können, muss besagte Klasse zum einen von *QObject* erben, darüber hinaus muss das Q_OBJECT Makro in der Klassen-Definition verwendet werden.

3.1.3 Slots

Slots sind Methoden, die im Rahmen des Event-Loops beim Auftreten von verbundenen Signalen, angestoßen werden. Auch diese müssen im Header der implementierenden Klasse speziell im Bereich "slots:" deklariert werden. Die Deklaration selbst entspricht der einer normalen Methode. Bei der Initialisierung der GUI-und $Modell$-Klasse werden die Signale mittels "connect" den entsprechenden Slots zugeordnet. Die Syntax lautet dabei

$$connect(SignalKlasse, SIGNAL(deklariertesSignal()),$$

$$SlotKlasse, SLOT(deklarierterSlot()));$$

Sollen vom Signal an den Slot Parameter übergeben werden, so werden nur deren Datentypen sowohl bei "deklariertesSignal()" als auch "deklarierterSlot()" als Parameter übergeben. Eine Benennung darf nicht erfolgen.

3.2 Modell-Klasse

Die *Modell*-Klasse (*ThesisModel*) kapselt jeglichen Zugriff auf Messdaten oder Operationen, die darauf ausgeführt werden und kann somit weitgehend unabhängig von der *GUI* betrieben werden. Der Konfigurations-Teil der Arbeit setzt jedoch ein gewisses Maß an Nutzer-Interaktion voraus, da dieser jeweils den *Transmitter* positionieren muss.

3.2.1 Kommunikation über die serielle Schnittstelle

Die Kommunikation über die serielle Schnittstelle wird im *masterthread*-Modul realisiert. Dieser Thread wird beim Verbinden mit der seriellen Schnittstelle gestartet und läuft über die gesamte Dauer der Programmausführung. Es gibt eine boolsche Variable, die der Pausierung des Pollings dient (bool *stopSerialPolling*) und eine weitere die den Modus festlegt (bool *createCoordinateSystemMode*). Das Pausieren des Threads wird über eine Mutual-Exclusion-Variable (mutex) realisiert, die den Thread in einen Wartezustand versetzt. Eine erneute Transaktion setzt die Ausführung fort. Bei einer solchen Transaktion wird die Anzahl der zu verwendenden Sensoren festgelegt. Die beiden verfügbaren Modi unterscheiden sich lediglich durch die emittierten Signale, die wiederum einen eigenen Slot in der *Modell*-Klasse auslösen.

3.2.2 Erstellung des Koordinatensystems

Zur Erstellung des Koordinatensystems wartet die *GUI* nach entsprechendem Befehl 10 Sekunden ab, bis die Koordinatensystem-Erstellung des *Modells* gestartet wird. In dieser Zeit soll der Nutzer den *Transmitter* zum entsprechenden *Sensor* bewegen. Danach werden für alle beteiligten *Sensoren*[7] jeweils 50 Messwerte gesammelt. Da die verwendeten *Sensoren* zwischen einem plausiblen Wert und Null schwanken ("flattern"), werden derartige Null-Werte verworfen, um eine möglichst präzise Messreihe zur Kalibrierung zu erhalten. Diese Messreihen werden für jeden *Sensor* durchgeführt, wobei jeweils nach Abschluss einer Messreihe und erneuter Betätigung der *GUI* 10 Sekunden verbleiben, um den *Transmitter* zum nächsten *Sensor* zu bringen.

Die Messungen werden im Einzelnen durch einen Slot im *Modell* verarbeitet, der vom *masterthread*, bei entsprechender Konfiguration[8], beim Eintreffen einer neuen Messung angestoßen wird.

[7] ausgeschlossen sind hierbei der *Sensor* von dem aus aktuell gemessen wird, sowie alle *Sensoren*, deren Index größer als die Maximal-Anzahl der zu verwendenden *Sensoren* ist

[8] die Variable *createCoordSysMode* = true

Nach Abschluss aller Messreihen werden die Mittelwerte der gemessenen Distanzen verwendet, um eine Matrix mit besagten Distanzen zu füllen, wobei die Elemente der Hauptdiagonale die Abstände eines *Sensors* zu sich selbst angeben und somit per Definition Null sind.

```
0 47 77 84 0
49 0 89 65 0
75 82 0 43 0
88 64 46 0 0
0 0 0 0 0
0 49 75 88 0
49 0 87 68 0
75 85 0 45 0
88 67 45 0 0
0 0 0 0 0
corVals: 1 1.04255 0.974026 1.04762 1
```

Abbildung 3.3: Screenshot der Konsolen-Ausgabe einer Distanz-Matrix vor und nach Normalisierung. Die Normalisierungsfaktoren sind unter corVals aufgeführt.

Bei näherer Betrachtung diser Matrix (3.3) fällt jedoch auf, dass zumeist die gemessene Distanz zwischen zwei *Sensoren* von der Richtung abhängt ($\overline{AB} = \overline{BA} \cdot \epsilon$). Zur Behebung dieses Problems werden die gemessenen Distanzen aller *Sensoren* auf die von *Sensor* 1 Gemessenen normalisiert und die entsprechenden Faktoren ϵ vermerkt. Diese Faktoren werden im späteren Verlauf immer auf die gemessenen Distanzen aufmultipliziert und finden auch bei der Berechnung des Koordinatensystems Anwendung. Die Abweichungen der Distanzen der einzelnen *Sensoren* betrugen bei bisherigen Messungen weniger als 5%.

Bei der Erstellung des Koordinatensystems wird *Sensor* 1 als Ursprung definiert. *Sensor* 2 gibt die Richtung der X-Achse vor. Die Höhe aller *Sensoren* ist gleich[9] und beträgt 2m. Die Positionen aller *Sensoren* liegen somit in einer Ebene, was die Berechnung durch die Reduktion auf zwei Dimensionen vereinfacht. Die Positionen der verbleibenden *Sensoren* werden zudem mit positiver Y-Koordinate angenommen. Je nach Anschluss der *Sensoren* am *Gateway*-Modul kann das Koordinatensystem folglich an der X-Achse gespiegelt sein.

Als Basis für die Berechnung der Koordinaten dient die zwei-dimensionale Kreisgleichung

$$x^2 + y^2 = r^2$$

Wobei x und y gesucht sind und r jeweils der gemessenen Distanz entspricht.

[9] im Idealfall, es gibt fertigungsbedingte Abweichungen, die sich aber unterhalb von einem cm bewegen

3.2.3 Positionsberechnung

Sind die Positionen der *Sensoren* einmal bekannt, so kann die relative Position des Objektes[10] im Raum anhand seiner Distanzen zu den bekannten *Sensor*-Positionen berechnet werden. Hierbei liegt die drei-dimensionale Kreisgleichung zugrunde:

$$x^2 + y^2 + z^2 = r^2$$

In der *Software* ist die Positionsberechnung durch einen Slot im *Modell* realisiert, der bei entsprechender Konfiguration[11] des *masterthreads* ausgelöst wird, sobald neue Messungen eintreffen. Der Programmablauf nach Abschluss der Konfigurations-Phase besteht im Wesentlichen aus folgenden Punkten:

1. Der *masterthread* leitet eine Messung ein, indem das *Gateway*-Modul per serieller Schnittstelle kommuniziert bekommt, welche *Sensoren* beteiligt werden sollen.

2. Der *masterthread* erhält vom *Gateway*-Modul die Messwerte und stößt den *Modell*-Slot an, der für die Positionsberechnung zuständig ist, dieser wiederrum emittiert ein Signal, um die *GUI* zu aktualisieren.

   An dieser Stelle wird ein weiteres Signal emittiert, das bislang mit keinem Slot verbunden ist, welches die relative Positionen der Sensoren und Objekte in Metern enthält. Dieses Signal ist als Schnittstelle für eine Einbindung der Lokalisierung gedacht.

[10] welches das *Transmitter*-Modul trägt
[11] die Variable *createCoordSysMode* = false

Kapitel 4

Mathematische Herleitung

4.1 Berechnung des Koordinatensystems (Ebene)

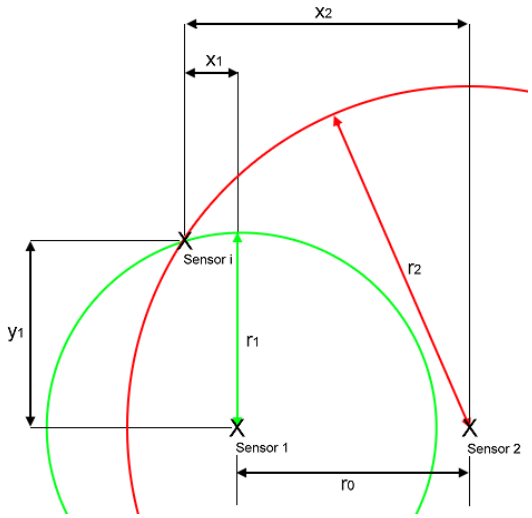

Abbildung 4.1: Anordnung der *Sensoren*

Wie an vorheriger Stelle bereits angemerkt, wird bei der Vermessung aller *Sensoren* S untereinander und dem daraus resultierenden Koordinatensystem vereinfachend angenommen, dass sich alle *Sensoren* in der gleichen Höhe befinden und dass sich der *Transmitter* immer unterhalb eben dieser befindet. Darüber hinaus definiert der an Position *eins* des *Gateways* angeschlossene *Sensor* den Koordinatenursprung und der *zweite* die X-Achse. Die Position der übrigen *Sensoren* wird in Relation zu besagten Beiden berechnet, wobei die Y-Koordinate als positiv ange-

nommen wird. Daraus folgen bereits die Positionen der ersten beiden *Sensoren* P_{S1} und P_{S2}:

$$P_{S1} = (0, 0)$$

$$P_{S2} = (\overline{S1S2}, 0)$$

Im Folgenden wird für jeden *Sensor* aus der Menge i (mit $i = S \setminus \{1, 2\}$) jeweils ein Kreis um *Sensor* 1 und *Sensor* 2 mit $r_0 = \overline{S1S2}$; $r_1 = \overline{S1Si}$; $r_2 = \overline{S2Si}$ geschlagen. Dabei seien x_j und y_j die Koordinaten des Kreises um *Sensor* j (für $j \in \{1, 2\}$). Im positiven Schnittpunkt der beiden Kreise befindet sich die Position des i-ten *Sensors* welcher durch gleichsetzen der beiden folgenden Gleichungen ermittelt wird:

$$(1) x_1^2 + y_1^2 = r_1^2$$

$$(2) x_2^2 + y_2^2 = r_2^2$$

Da die ersten beiden *Sensoren* die X-Achse definieren gilt:

$$y_1 = y_2$$

Darüber hinaus kann man x_1 in x_2 umrechnen durch:

$$x_2 = x_1 - r_0$$

dies in (2) eingesetzt liefert:

$$(1) x_1^2 + y_1^2 = r_1^2$$

$$(3) (x_1 - r_0)^2 + y_1^2 = r_2^2$$

Stellt man die Gleichungen 1 und 3 jetzt um und setzt sie gleich, erhält man:

$$(x_1 - r_0)^2 - r_2^2 = x_1^2 - r_1^2$$

Diese Gleichung liefert umgestellt nach x_1

$$x_1 = \frac{r_1^2 - r_2^2 + r_0^2}{2 \cdot r_0}$$

y_1 erhält man durch einsetzen von x_1 in (1)

$$y_1 = |\sqrt{r_1^2 - x_1^2}|$$

Da *Sensor* 1 im Ursprung liegt kann nun für jeden der i *Sensoren* die Position P_{Si} gesetzt werden:

$$P_{Si} = (x_1, y_1)$$

4.2 Positionsberechnung (Raum)

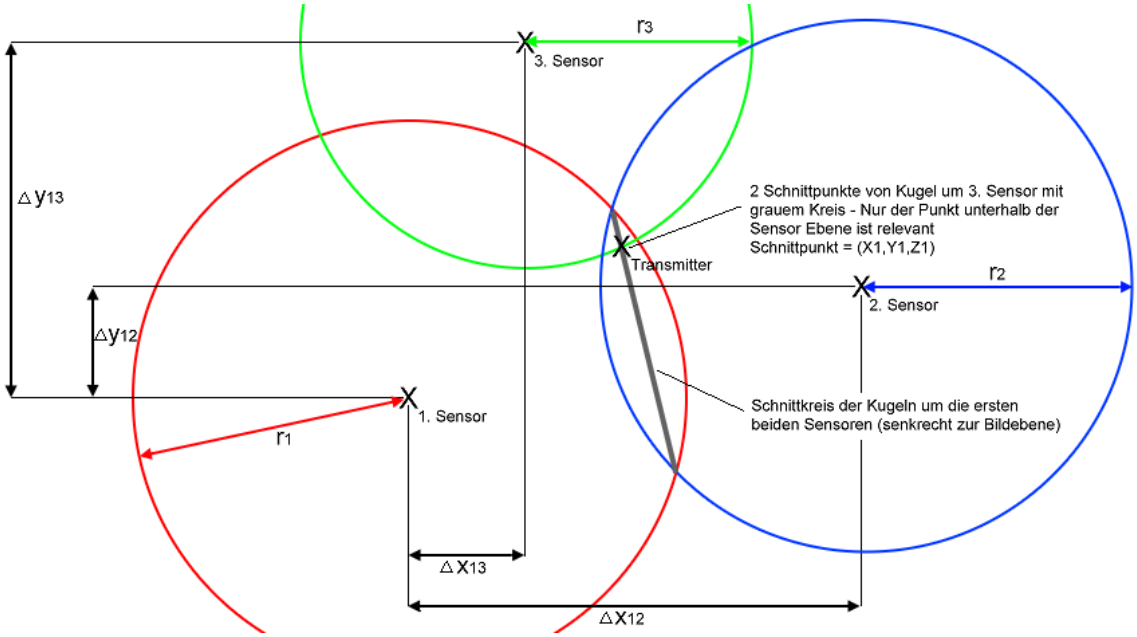

Abbildung 4.2: Position des *Transmitters* in gegebenem Koordinatensystem. r_1, r_2 und r_3 sind die Messwerte. Die Umrechnung von z.B. x_2 zu x_1 erfolgt analog zu der Umrechnung aus der Berechnung des Koordinatensystems

Nachdem nun die Positionen der *Sensoren* bekannt sind, können jeweils drei von diesen verwendet werden, um mittels der gemessenen Distanz zum *Transmitter* dessen Position zu ermitteln. Im Folgenden bezeichnet Δx_{ij} die Differenz der X-Koordinaten der *Sensoren* i und j. Selbiges gilt für die Y-Koordinaten. Zur Berechnung der Position im Raum wird die drei dimensionale Kreisgleichung verwendet:

$$x_i^2 + y_i^2 + z_i^2 = r_i^2$$

Für jeweils drei *Sensoren* erhält man nun folgende Gleichungen, wobei x_2 und x_3 respektive y_2 und y_3 durch x_1 und y_1 wie im vorherigen Abschnitt ersetzt wurden.

$$(4) x_1^2 + y_1^2 + z_1^2 = r_1^2$$

$$(5) (x_1 - \Delta x_{12})^2 + (y_1 - \Delta y_{12})^2 + z_1^2 = r_2^2$$

$$(6) (x_1 - \Delta x_{13})^2 + (y_1 - \Delta y_{13})^2 + z_1^2 = r_3^2$$

(5)-(4) liefert nun

$$(7) - 2x_1 \cdot \Delta x_{12} + \Delta x_{12}^2 - 2y_1 \cdot \Delta y_{12} + \Delta y_{12}^2 = r_2^2 - r_1^2$$

und (6)-(4) liefert

$$(8) - 2x_1 \cdot \Delta x_{13} + \Delta x_{13}^2 - 2y_1 \cdot \Delta y_{13} + \Delta y_{13}^2 = r_3^2 - r_1^2$$

stellt man nun (7) nach x_1 frei erhält man:

$$x_1 = \frac{r_2^2 - r_1^2 - \Delta y_{12}^2 + 2y_1 \cdot \Delta y_{12} - \Delta x_{12}^2}{-2\Delta x_{12}}$$

setzt man nun x_1 in (8) ein erhält man:

$$(\frac{r_2^2 - r_1^2 - \Delta y_{12}^2 + 2y_1 \cdot \Delta y_{12} - \Delta x_{12}^2}{\Delta x_{12}}) \cdot \Delta x_{13} + \Delta x_{13}^2 - 2y_1 \cdot \Delta y_{13} + \Delta y_{13}^2 = r_3^2 - r_1^2$$

umgestellt erhält man

$$(r_2^2 - r_1^2 - \Delta y_{12}^2 - \Delta x_{12}^2) \cdot \frac{\Delta x_{13}}{\Delta x_{12}} + \frac{2y_1 \cdot \Delta y_{12} \cdot \Delta x_{13}}{\Delta x_{12}} + \Delta x_{13}^2 - 2y_1 \cdot \Delta y_{13} + \Delta y_{13}^2 = r_3^2 - r_1^2$$

im folgenden sei

$$(r_2^2 - r_1^2 - \Delta y_{12}^2 - \Delta x_{12}^2) \cdot \frac{\Delta x_{13}}{\Delta x_{12}} = \alpha$$

damit

$$\alpha + \frac{\Delta x_{13}^2 + \Delta y_{13}^2 - r_3^2 + r_1^2}{2} = y_1 \cdot (\Delta y_{13} - \frac{\Delta y_{12} \cdot \Delta x_{13}}{\Delta x_{12}})$$

mit

$$\Delta y_{13} - \frac{\Delta y_{12} \cdot \Delta x_{13}}{\Delta x_{12}} = \beta$$

$$y_1 = \frac{\alpha + \frac{\Delta x_{13}^2 + \Delta y_{13}^2 - r_3^2 + r_1^2}{2}}{\beta}$$

Mit bekannten x_1 und y_1 lässt sich z_1 leicht berechnen durch:

$$z_1 = \sqrt{r_1^2 - x_1^2 - y_1^2}$$

4.3 Überlegungen zur Erhöhung der numerischen Stabilität

Bei näherer Betrachtung der Gleichung zur Berechnung von x_1 fällt auf, dass die Differenz der X-Koordinaten der ersten zwei gewählten *Sensoren* im Nenner vorkommt. Je nach Wahl der Reihenfolge der *Sensoren* kann es nun vorkommen, dass besagte Differenz relativ gering ausfällt, wodurch Rundungs- und Mess-Fehler

deutlich ins Gewicht fallen. Der hier potentiell falsche errechnete Wert für x_1 fließt zudem in die Berechnung der Werte für y_1 und z_1 ein, sodass sich der Fehler fortpflanzt. Da die Reihenfolge der *Sensoren* willkürlich gewählt ist, verwendet die *Software* an dieser Stelle als zweiten *Sensor* denjenigen, der den größten X-Koordinaten-Abstand zum ersten *Sensor* hat. Eine weitere Problematik stellt die Wurzel in der Berechnung der Höhe da, da deren Argument bei geringen Höhen-Differenzen, also bei einer *Transmitter*-Position von wenigen Zentimetern unterhalb der der *Sensoren*[1], leicht negativ werden kann.

4.4 Berechnung des Skalierungsfaktors

Die mit 14 MHz getakteten *Sensoren* verwenden einen 16-Bit-Timer, um zu messen, wie lange es dauert, bis der Schall vom *Transmitter* beim jeweiligen *Sensor* ankommt. Der Prescaler des Timers ist $\frac{1}{8}$. Das Ergebnis einer Distanz-Messung ist das High-Byte des Timers[2]. Der Takt des Timers ist folglich:

$$TIMER\_CLOCK = MCU\_CLOCK \cdot TIMER\_PRESCALER$$
$$= \frac{14 \cdot 10^6}{8} \frac{1}{s}$$

Die Distanz *dist* ist das High-Byte des Timers für eine Distanz-Messung. Der Timer hat also in etwa den Wert

$$TIMER\_VALUE = dist \cdot 2^8 = dist \cdot 256$$

Teilt man nun den Timer-Wert durch die Timer-Taktrate erhält man die "Flugzeit" des Schalls für die gegebene Distanz. Multipliziert man diese mit der Schallgeschwindigkeit erhält man die Distanz in Metern.

$$dist[m] = \frac{TIMER\_VALUE}{TIMER\_CLOCK} \cdot v_{sound}$$

Klammert man nun die gemessene Distanz *dist* aus, erhält man:

$$dist[m] = dist \cdot \frac{256 \cdot v_{sound}}{TIMER\_CLOCK}$$

mit dem Umrechnungs-Faktor

$$corr\_fac = \frac{256 \cdot v_{sound}}{TIMER\_CLOCK}$$

[1] je größer die Testperson ist, desto eher kann dies passieren
[2] das Low-Byte ist verrauscht

Kapitel 5

Betrieb

5.1 Inbetriebnahme

1. Drei bis fünf *Sensoren* im Raum aufstellen. Die Ultraschall-Empfänger der HC-SR04-Module sollten möglichst gleichermaßen auf alle anderen *Sensoren* gerichtet sein (jeder *Sensor* sollte jeden anderen *Sensor* "hören" können).

2. Die *Sensoren* mit Strom versorgen.

3. *Gateway* und *Transmitter* mit Strom versorgen. Es darf immer nur ein *Transmitter* aktiv, also unter Strom, sein. Für die Kalibrierungs-Phase wird die kabelgebundene Version empfohlen.

4. *Gateway*, *Sensoren* und *Transmitter* mit Ethernet-Kabeln verbinden. Die Kabel der *Sensoren* so verlegen, dass das zu lokalisierende Objekt nicht dadurch beinträchtigt wird. Das Kabel des *Transmitters* muss lang genug sein, um jeden *Sensor* erreichen zu können. Der *Transmitter* wird an dem RJ45-Stecker des *Gateways* eingesteckt, der mit gelben Litzen mit dem *Gateway* verbunden ist. Der Stecker, der am zweit-nächsten am Mikrocontroller des *Gateways* liegt, ist für den 1. *Sensor*, also den Ursprung des Koordinaten-Systems, vorgesehen. Der nächste *Sensor* definiert die X-Achse. Die übrigen *Sensoren* sollten nun oberhalb der X-Achse liegen[1]. Die Reihenfolge ist bei besagten *Sensoren* beliebig, diese muss aber in der Kalibrierungs-Phase entsprechend abgelaufen werden, es empfielt sich also eine reproduzierbare Anordnung (z.B. gegen den Uhrzeigersinn).

5. Die *Software* starten, den richtigen COM-Port auswählen und Connect klicken. Der *Transmitter* muss an dieser Stelle einmal klicken.

[1] bei einem Koordinaten-System mit der X-Achse nach rechts verlaufend, wären diese *Sensoren* "oben" respektive "vorne"

6. Die Anzahl der zu verwendenen *Sensoren* auf die aufgestellte Anzahl einstellen (Standardmäßig vier).

7. Die korrekte Funktion aller *Sensoren* testen: Start klicken und für jeden *Sensor* den *Transmitter* auf den betreffenden *Sensor* hinzu und weg bewegen[2]. Der Balken des entsprechenden *Sensors* muss sich proportional zur Bewegung des *Transmitters* verschieben. Der *Transmitter* muss bei jeder Messung klicken.

8. Die Test-Phase durch klicken von Stop stoppen. Die Kalibrierungs-Phase mit "Create for Sensor 1/n" [3] starten. Den *Transmitter* zum ersten *Sensor* bewegen, diesen dort dicht oberhalb des besagten *Sensors* halten, bis die Messreihe abgeschlossen ist[4].

9. Die Kalibrierungs-Phase durch wiederholtes klicken von "Create for Sensor x/n" fortsetzen und abschließen. Wie zuvor muss der *Transmitter* zum x-ten *Sensor* gebracht und eine Messreihe abgeschlossen werden. Sobald alle n *Sensoren* vermessen wurden, ist die Kalibrierung abgeschlossen. Das Status-Label sollte dies ausgeben und die Grafik im unteren Bereich sollte die *Sensor*-Positionen in der X-Y-Ebene anzeigen.

10. Positionsmessungen durch klicken von Start starten.

5.2 Häufige Fehlerursachen beheben

1. Bei einer Messungen muss ein *Transmitter* im Bereich der Ultraschall-Emitter klicken, ist dies nicht der Fall, ist womöglich ein Kabel an den Ultraschall-Emittern oder den RJ45-Buchsen abgerissen.

2. Die Kontakte des Step-Up-Wandlers sollten nicht blank liegen, da der Mikrocontroller nicht in Kontakt mit 12V kommen sollte[5]. Die Isolierung entsprechend in Stand halten.

3. Verhält sich ein *Sensor* in seiner direkten Umgebung (etwa 1,2m) normal und darüber hinaus nicht, so ist die Spannungsversorgung unzureichend[6]. Diese entsprechend tauschen.

---

[2] dabei muss eine Distanz von 1,2m zum *Sensor* überschritten werden!

[3] bei maximal n *Sensoren*

[4] der *Transmitter* klickt nicht mehr

[5] dieser muss sonst evtl. getauscht werden - lFuse = $0xE4$, hFuse = $0xD9$, Transmitter.hex flashen

[6] siehe Kapitel: Aufbau der Module - Abschnitt Sensoren

4. Sollte ein *Sensor* keine plausiblen Daten liefern, prüfen, ob die RJ45 Buchsen auf beiden Seiten komplett verbunden sind. Ist dies der Fall, ist vermutlich die Konstruktion der HC-SR04-Module schadhaft. Dabei reißen oft die Masse, VCC oder Trigger Leitungen mindestens eines HC-SR04-Moduls von dem Verbund an den verzinkten Kupferleitungen ab. Diese dann nachlöten.

5. In seltenen Fällen kann ein USB zu Seriell Wandler Probleme verursachen[7], diesen dann aus- und wieder einstecken. Eine vorher erfolgte Kalibrierung kann dadurch nichtig gemacht werden. Geht ein Rechner in den Ruhemodus, kann es sein, dass das gerade genannte Phänomen danach auftritt.

6. In der Kalibrierungs-Phase kann viel Geduld von Nöten sein. Es könnte eine zweite Person hilfreich sein, die prüft, welcher *Sensor* noch nicht genügend gültige Messwerte gesammelt hat[8]. Womöglich sollte dieser *Sensor* temporär in die Richtung des *Transmitters* gedreht werden[9].

[7] unplausible Daten senden oder empfangen

[8] Es gibt diesbezüglich eine Konsolenausgabe, die aber aus der Distanz nicht unbedingt lesbar ist.

[9] dabei aufpassen, dass der *Sensor* nur gedreht und nicht verschoben wird! Den *Sensor* anschließend wieder zurück drehen.

Kapitel 6

Fazit

6.1 Präzision

In Tests konnte ein Versuchsaufbau mit vier *Sensoren* auf einer Fläche von etwa 4 × 6 m weitgehend stabil betrieben werden, dabei wurden die gemessenen Distanzen jeweils mit 7 Bit aufgelöst, was zu einer Präzision von etwa $\pm 2,4$ cm pro Distanzmessung von *Transmitter* zu einem *Sensor* führen dürfte. Vor allem in den Randbereichen und in der Höhe treten die größten Ungenauigkeiten auf. Dies liegt zum Einen vermutlich daran, dass die Höhe die letzte der drei berechneten Koordinaten ist, wo sich somit die Fehler akkumulieren. Des Weiteren sinkt die Präzision, je näher man der Höhe der *Sensoren* kommt, denn hier werden die Höhendifferenzen klein und die Fehler[1] haben einen deutlicheren Einfluss auf das Ergebnis. Dieser Effekt könnte durch eine deutlich höhere Platzierung der *Sensoren* gemildert werden.

Die *Senoren* wurden bislang nicht auf dem Fußboden montiert, da ansonsten zu viele Hindernisse[2] die Ultraschall-Messungen stören könnten.

Die Präzision der Bewegung in der X-Y-Ebene ließe sich durch mitteln der einzelnen Koordinaten (bei mehr als drei *Sensoren*) auf wenige Zentimeter genau erfassen, was den Anforderungen einer Virtual-Reality-Anwendungen genügen könnte, ab nicht getestet wurde. Die Präzision der Höhe dürfte dem jedoch nicht genügen. Die Höhendaten müssten vermutlich deutlich geglättet und verbessert werden[3]. Hierzu könnten weitere Sensoren[4] verwendet werden, wobei die Fusion der unterschiedlichen Sensor-Daten[5] das sinnvollste Vorgehen dastellen dürfte.

[1] Numerisch oder Messungungenauigkeit
[2] vor allem die Person, die den *Transmitter* trägt
[3] womöglich würde das Mitteln über alle Triplet-Punkte bereits Besserung schaffen
[4] z.B. ein Inertial-Sensor
[5] z.B. mit Hilfe eines Kalman-Filters

6.2 Technische Stabilität

Hardware bedingte Ausfälle aufgrund mechanischer Belastung sind keine Seltenheit. So reißen zum Teil die dünnen Kabel der RJ45-Buchsen ab, die von den starren Ethernet-Kabeln gebogen und geknickt werden. Auch die mechanische Belastung des *Transmitters* kann, aufgrund seiner ständigen Bewegung, abgerissene Kabel am Ultraschall-Sender zur Folge haben.

Die *Sensoren* sind durch die Befestigung an den Ständern besser vor abreißenden Kabeln geschützt, jedoch stellen hierbei die Ständer ein Problem hinsichtlich des Transports da. Zum Einen kann man bei 2 m langen Stangen leicht mit den empfindlichen *Sensoren* an einer Seite irgendwo anstoßen, zum Anderen ist eine Montage/Demontage der Sensoren auf den Stangen auch oft mit einer Beschädigung verbunden. Darüber hinaus können die Stangen im Betrieb umgerissen werden, wenn der Probant dagegen rennt oder an einem Ethernet-Kabel hängen bleibt.

Wie bereits im Hardware-Kapitel erwähnt, sind die *Sensoren* überaus anfällig hinsichtlich der Stromquelle. Das Phänomen äußert sich dadurch, dass der betreffende *Sensor* in seiner Nähe[6] plausible Werte liefert, aber weiterhin um eine Distanz von 120 cm schwankt, sobald man sich weiter entfernt.

Aufgrund dieser Probleme empfielt sich dringend, nach einem Versuchsaufbau und vor der Kalibrierungs-Phase zu überprüfen, ob alle *Sensoren* und *Transmitter* funktionieren. Dazu die *Software* starten, *connect* und *start* klicken. Jetzt jeden einzelnen *Sensor* betrachten[7] und den *Transmitter* bewegen. Die Messwerte müssen sich nun passend zur Bewegung ändern - dies auch mit verschiedenen Winkeln zum jeweiligen *Sensor* probieren. Dabei auch prüfen, ob jeder *Sensor* jeden Anderen "hören" kann.

6.3 Ausblick

Ultraschall ist heutzutage nicht mehr unbedingt Forschungsgegenstand, obwohl sich damit erstaunlich präzise Ergebnisse erzielen lassen[Jim05]. Auch im Rahmen einer weniger professionellen Lösung, wie beispielsweise dieser Abschlussarbeit, können kostengünstig[8] durchaus interessante Ergebnisse erreicht werden, jedoch ist fraglich, ob die Präzision für den Anwendungsfall "Virtual Reality" ausreicht. Auch die Notwendigkeit, außer der Brille noch weitere Ausrüstung auf dem Kopf herumtragen zu müssen (die ständig klickt), könnte sich als zu lästig herausstellen.

[6] bis zu einer Entfernung von etwa 120 cm

[7] genauer: die Distanz-Balken, zur Not das Bild vergrößern

[8] die Gesamtkosten dürften sich zwischen 100 und 200 Euro bewegen

Dennoch könnte das Ergebnis dieser Arbeit durch Hinzunahme weiterer Sensoren[9] vermutlich verbessert werden. Auch bestünde die Möglichkeit, jedes Ultraschall-Modul mit einem einzelnen Mikrocontroller auszustatten und somit das "flattern" zu eliminieren. Es ist jedoch überaus fraglich, ob sich der höhere Aufwand[10] auszahlt. Darüber hinaus wäre eine eigene Konstruktion der Ultraschall-Empfänger womöglich wünschenswert was jedoch einige Kenntnisse der Elektrotechnik vorraus setzt.

Eine interessante Alternative ließe sich vermutlich mit Hilfe der Bildverarbeitung basierend auf Kameradaten realisieren. Dies wäre hinsichtlich des Budgets sicher teurer[11] als die Ultraschall-Lokalisierung, dürfte aber deutlich unterhalb der Kosten für eine Laser-gestützte Positionsermittlung liegen[12].

Inzwischen erscheinen auch Fortschritte im Bereich der Funk-gestützen-Ortung als viel versprechend. So sehen dies zumindest die Konstrukteure des US-Unternehmens The Void, die bereits gute Ergebnisse durch Kamera-basierte-Lokalisierung erzielen und diese durch den Umstieg auf Funk weiter verbessern wollen [Jan15] [Voi15].

Die Komplexität einer derartigen Arbeit könnte aber vermutlich schnell den Umfang einer Studien- oder Abschluss-Arbeit überschreiten. Bei einem weniger professionellen Vorgehen sinkt in der Regel auch wieder die Präzision.

[9] vgl Abschnitt Präzision
[10] Kosten, Konstruktion und Synchronisierung
[11] grob geschätzt: 1000 Euro für Kameras und Material
[12] mindestens 5000 Euro für einen 2D-Laserscanner

Abbildungsverzeichnis

Literaturverzeichnis

[Jan15] Janssen, Jan-Keno: *Pixel zum Anfassen - Ein Besuch im Virtual-Reality-Center The Void*. Website, 2015. – Online erhältlich unter http://www.heise.de/ct/ausgabe/2015-26-Ein-Besuch-im-Virtual-Reality-Center-The-Void-3015429.html; abgerufen am 3. Januar 2016.

[Jim05] Jiménez, A.R.: *Ultrasonic Localization Methods for Accurate Positioning*. 2005

[K.07] K., Benedikt: *Homepage*. Website, 2007. – Online erhältlich unter http://www.mikrocontroller.net/topic/67273; abgerufen am 23. Januar 2016.

[Mis13] Mischnick, Stephan: *Homepage*. Website, 2013. – Online erhältlich unter http://www.strippenstrolch.de/1-2-19-motortreiber-l6203-erkunden.html; abgerufen am 9. Januar 2016.

[Plc13] Plc, Digia: *Homepage*. Website, 2013. – Online erhältlich unter http://doc.qt.io/qt-5/qtwidgets-painting-basicdrawing-example.html; abgerufen am 27. Januar 2016.

[Shi12] Shienkov, Denis: *Homepage*. Website, 2012. – Online erhältlich unter http://doc.qt.io/qt-5/qtserialport-blockingmaster-example.html; abgerufen am 27. Januar 2016.

[Voi15] Void, Webmaster T.: *Homepage*. Website, 2015. – Online erhältlich unter https://thevoid.com/; abgerufen am 3. Januar 2016.

Anhang

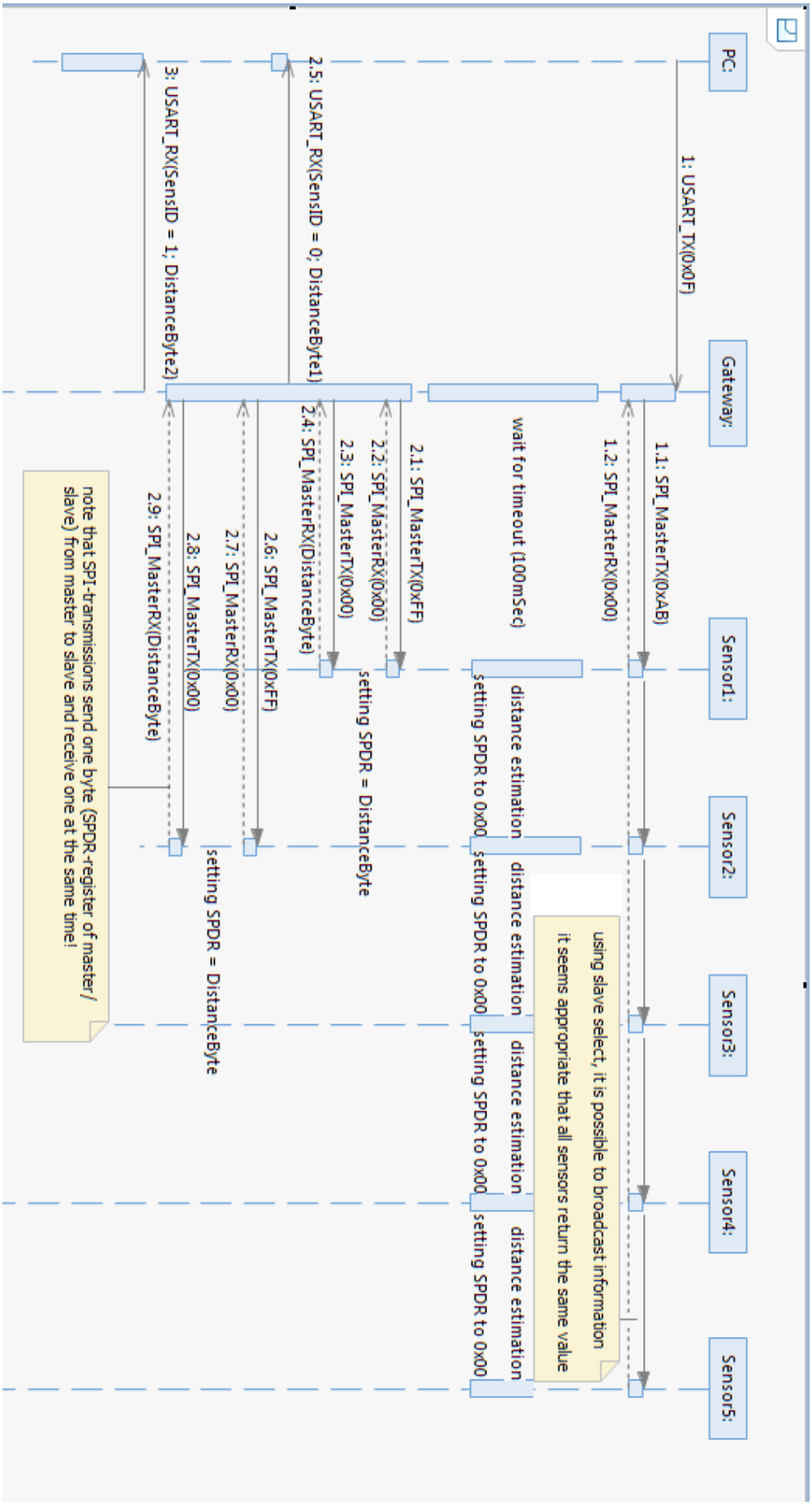

Abbildung 1: vergrößerte Grafik aus Kapitel Aufbau der Module, Abschnitt Kommunikation