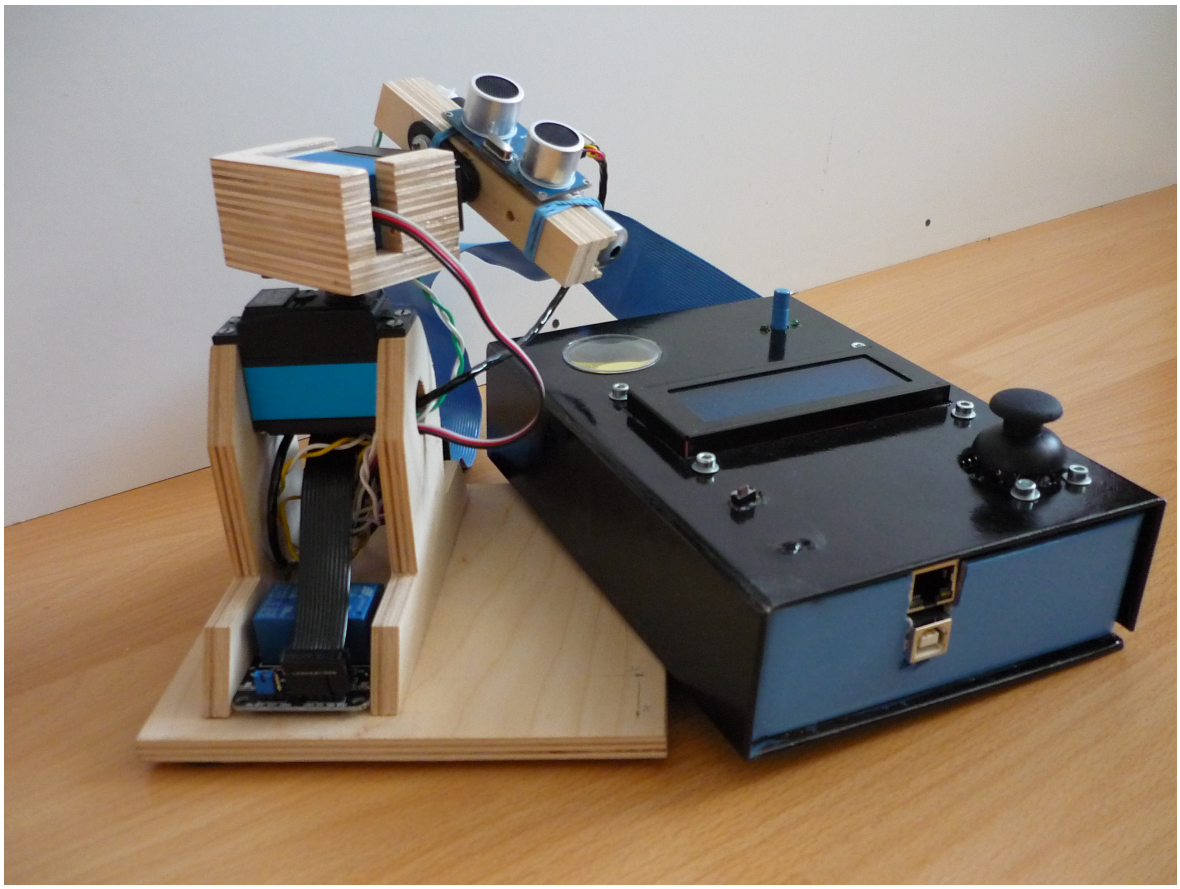
JUGEND FORSCHT

KONRAD MERKEL

Mathematische Beschreibung und technische Umsetzung eines Multifunktionsgerätes mithilfe eines Mikrocontrollers und systemübergreifender Programmierung



Autor: Konrad Merkel
Abgabedatum: 10. März 2014

Inhaltsverzeichnis

0.1. Danksagung

Bevor ich mein Projekt erläutere, möchte ich all denen danken, die mir bei der technischen Umsetzung geholfen oder mir Geräte zur Verfügung gestellt haben. Einen besonderen Dank gebührt dabei meinem ehemaligen Technik & Computer-Lehrer Gerd Schulze. Durch ihn war es mir möglich, die Holzkomponenten des entwickelten Gerätes mithilfe einer CNC-Maschine herzustellen.

Des weiteren möchte ich mich auch bei meinen Eltern bedanken, die mich finanziell unterstützt und gefördert haben. Ohne ihre Hilfe stünde ich heute nicht da wo ich jetzt bin. Dafür bin ich ihnen sehr dankbar.

0.2. Einleitung

Jeder kennt die Probleme, wenn man eine Reihe von Bildern an einer Wand aufhängen möchte. Sie sollten mittig hängen und vor allem nicht schief. Eine einfache Alternative zu den aufwendigen Vermessungen bietet das hier beschriebene Gerät namens Moskito. Man stellt es parallel zur Wand, peilt zwei sich gegenüberliegende Eckpunkt an und kann auf einer benutzerfreundlichen Bedienoberfläche sofort die Bohrpunkte markieren lassen, die man möchte. Die komplexen mathematischen Berechnungen finden dazu auf dem verwendeten Computer statt. Das Gerät kann nicht nur Wände vermessen, sondern auch viele weitere Aufgaben erfüllen. Dafür hat es eine spezielle Softwarearchitektur, die einen Multifunktionseinsatzt möglich machen.

Der Moskito besteht aus zwei Servomotoren, vielen elektronischen Bauelementen und einem Laser, der dann auf jeden beliebigen Punkt im Raum oder an der Wand zeigen kann. Gesteuert wird das Ganze mit einem Mikrocontroller und verschiedenen Modulen, wie einem Relaismodul, einem LCD-Display, einem Ethernetshild und eigens entwickelten Schaltkreisen. Trotz Komplexität der Steuerung ist ein vielfältiger Einsatz möglich.

Diese Dokumentation ist in zwei größere Abschnitte gegliedert. Zunächst werden die mathematischen Modelle, Gleichungen und Formeln für die Berechnung der Zielerfassung erläutert. Sie bilden die Grundlage für das Funktionieren in der späteren Anwendung und waren auch der Anfang des Projektes. Im zweiten großen Teil dieser Arbeit gehe ich auf die technische Umsetzung und die mehrschichtige und plattformübergreifende Softwarearchitektur ein.

0.3. Geschichte der Entstehung

Den Anfang des Projektes bildete eine mathematische Überlegung. Durch Zufall sah ich an einem Spielzeughubschrauber meines Bruders ein frei bewegliches Kamerasystem. Ich dachte mir die Piloten des Helikopters müssten aus ihrem Cockpit heraus die Kamera steuern und Ziele anvisieren können. Das Problem war: Die Drehachsen der Kamera waren durch zwei Gelenke und Abständen der Gelenke zueinander nicht übereinander. Die Winkel beeinflussten sich gegenseitig, was eine einfache Berechnung der Winkel unmöglich macht. Ab diesem Zeitpunkt faszinierte es mich eine Rechenvorschrift zu finden um mit der Kamera auf bestimmte Gegenstände zu zielen. Insgesamt brauchte ich über eine Woche, unzählige Blätter Papier und viele Nerven um einen Rechenansatz zu finden.

Später beschloss ich aus der reinen theoretischen Berechnung eine praktische Anwendung zu schaffen. Dazu verwendete ich zwei Servomotoren, die ich übereinander montierte. Die Drehachsen sind nicht genau übereinander und im rechten Winkel zueinander. Das heißt auch hier kann eine einfache Bestimmung der Drehwinkel nicht vorgenommen werden und es muss zwangsläufig auf das theoretische Modell zurückgegriffen werden.

Im Laufe der Zeit verbesserte ich das entstandene Gerät und schrieb eine passende Software für jede Anwendung.

0.4. Zusammenfassung

Moskito bezeichnet ein Multifunktionsgerät bestehend aus zwei Servomotoren, einem Laser und weiteren Sensoren und Aktoren. Gesteuert wird die Hardware durch einen Mikrocontroller. Das Gerät kann somit zum Beispiel als Vermessungs- und Markierungssystem für Hand- und Heimwerker werden.

Das besondere an diesem Projekt ist, dass von der Idee über die Umsetzung bis hin zum fertigen Produkt die gesamte Entwicklung in einer Hand liegt. Das betrifft

- die Mathematik,
- die Fertigung der Holzkomponenten,
- die Schaltkreise,
- die Programmierung auf mehreren Plattformen in mehreren Schichten (layered system structure) und
- ausführlichen Tests.

Für viele Anwendungsgebiete des Moskito, wie zum Beispiel als Ziel- und Vermessungssystem, benötigt man eine Reihe von Gleichungen und Formel, die die Koordinaten in Drehwinkel der Servos umrechnen können. Ich nutze dazu ein abstraktes mathematisches Modell auf Basis der Analysis. Die entsprechenden Formeln sind im ersten Gliederungspunkt genau erläutert.

Die Hard- und Software ist in drei wesentliche Teil, Ebenen unterteilt. Als erste Ebene bildet den Kern das universale Betriebssystem des Mikrocontrollers der für die Steuerung der Hardwarekomponenten verantwortlich ist. Dieses System ist modular aufgebaut. Dabei hat jedes Modul eine bestimmte Aufgabe, die es erfüllen muss. Je nach Wichtigkeit dieser Aufgabe wird dem Modul entsprechend mehr oder weniger CPU-Zeit gegeben.

Die zweite Ebene ist die Anwenderschicht auf einem Computer, der mit dem Gerät über ein USB-Kabel verbunden ist. Dieser Teil bietet eine hohe Flexibilität und Benutzerfreundlichkeit, da es zum einen grafische Benutzeroberflächen gibt und zum anderen für jeden Einsatz das entsprechende und spezialisierte Programm. Durch eine entsprechende Entwicklerschnittstelle am Computer können auch fremde Entwickler ohne viel Aufwand das Projekt erweitern. Das heißt das eine Spezialisierung durch adaptive Anwenderprogramme schnell und effizient erfolgen kann. Aus diesem Grund bezeichnet man den Moskito als Multifunktionsgerät.

Die dritte Ebene ist die Monitorring-Schicht. Ist der Mikrocontroller an ein Netzwerk angeschlossen, so ist er in der Lage sich per DHCP mit dem Netzwerk zu verbinden und stellt dann eine Webseite zur Verfügung. Auf dieser Webseite gibt es einen geschützten Bereich, auf dem die Positionen der Servomotoren und der Laserstatus angezeigt werden.

1. Mathematische Modellierung

Ziel der Modellierung ist es Formeln zu finden, die Zielkoordinaten in Drehwinkel der Servomotoren umrechnen. Der horizontale Winkel wird als Alpha und der vertikale Winkel als Beta bezeichnet. Im folgenden wird die Herleitung einer geeigneten Formel erläutert.

Die Wahl eines geeigneten Bezugssystems Um die Formeln nicht unnötig zu erschweren, lege ich fest, dass das Gerät sein eigenes kartesisches Koordinatensystem aufspannt. Der Koordinatenursprung liegt dabei auf der horizontalen Drehachse am unteren Rand des Ständers. Die x- und y-Achsen sind entsprechend der Beschriftung auf dem Gerät vermerkt. Sollte man ein anderes Koordinatensystem verwenden, so muss man die Koordinaten umrechnen und kann danach die selben Formeln verwenden.

1.1. Berechnung des horizontalen Drehwinkels (Alpha)

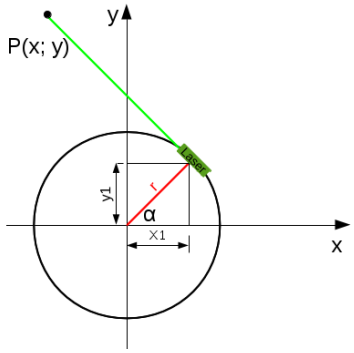


Abbildung 1.1.: Skizze für die Alphaberechnung

Die Berechnung von Alpha stellte sich als äußerst umfangreich heraus. Das liegt vor allem daran, dass der Laser nicht über der horizontalen Drehachse liegt, sondern bei Drehung einen Kreisbogen um diese fährt. Betrachtet man die Konstruktion von oben, so bildet der Laserstrahl eine Tangente an den Kreisbogen, den der horizontale Servomotor abfährt. Die Formel für eine Tangente am Kreis ist der Ausgangspunkt für die Herleitung. Der Mittelpunkt des Kreises befindet sich dazu über dem Koordinatenursprung. Die Z-Koordinate (senkrechte Höhe) ist für den Alpha-Winkel nicht von Bedeutung. Deswegen kann, durch einen Blick von oben, das Modell gemäß nachfolgender Gleichung ins zweidimensionale projiziert werden.

$$xx_1 + yy_1 = r^2 \tag{1.1}$$

x_1 und y_1 sind dabei die gemeinsamen Koordinaten der Tangente und des Kreises. x und y bilden Punkte auf der Geraden bzw. die Koordinaten des Punktes, auf den gezielt werden soll. Der Radius des Kreises r ist von der Konstruktion der Vorrichtung abhängig und wurde durch Messung bestimmt.

Der Drehwinkel α entspricht dem Winkel zwischen x-Achse und der Normalen im Punkt $P(x_1; y_1)$. Entsprechend der Winkelbeziehungen ergibt sich:

$$y_1 = r \cdot \sin\alpha \tag{1.2}$$

$$x_1 = r \cdot \cos\alpha \tag{1.3}$$

Setzt man nun beide Formeln in die Gleichung 1.1 ein, so erhält man eine Formel für die Berechnung des horizontalen Drehwinkels (x und y als Koordinaten des Zielpunktes).

$$r = x \cdot \cos\alpha + y \cdot \sin\alpha \tag{1.4}$$

Umgestellt nach α ergeben sich folgende Gleichungen:

$$\alpha = 2\tan^{-1}\frac{y+\sqrt{x^2+y^2-r^2}}{x+r} \tag{1.5}$$

$$\alpha = 2\tan^{-1}\frac{y-\sqrt{x^2+y^2-r^2}}{x+r} \tag{1.6}$$

Die zwei Gleichungen zeigen zwei unterschiedliche Lösungen. Zu jeder Gleichung gibt es noch unendlich viele Lösungen mehr, da man die Lösungen jeweils mit $360\,^\circ$ addieren kann. Es sind aber nur die kleinsten Ergebnisse von Bedeutung. Von praktischer Bedeutung ist nur eine Gleichung, da die Servomotoren sich jeweils nur bis $180\,^\circ$ drehen können.

1.2. Berechnung des vertikalen Drehwinkels (Beta)

Die Berechnung des Beta-Winkels ergibt sich aus Dreiecksbeziehungen (siehe Skizzen 1.2). Aufgrund des kartesischen Koordinatensystems sind die Dreiecke rechtwinklig. In den folgenden Gleichungen steht h jeweils für die Höhe, in der sich der Laser befindet oder anders ausgedrückt die z-Koordinate des Lasers. x, y und z stehen für die Koordinaten des Zielpunktes P. β ist der obere Winkel zwischen z-Achse und Laser.

Für alle Punkte, bei denen $z < h$ gilt, wird Beta mithilfe folgender Gleichung berechnet:

$$\tan(180-\beta) = \frac{\sqrt{x^2+y^2}}{h-z} \tag{1.7}$$

Im Zähler steht dabei der Abstand des Punktes zur z-Achse. Bei allen Punkten, bei denen $z > h$ gilt, wird der vertikale Drehwinkel durch folgende Gleichung errechnet:

$$\tan\beta = \frac{\sqrt{x^2+y^2}}{z-h} \tag{1.8}$$

Sollten die Höhe und die z-Koordinate gleich sein, so ist $\beta = 90\,^\circ$

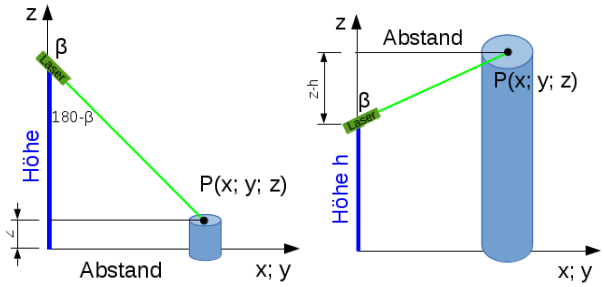


Abbildung 1.2.: Skizzen für die Betaberechnung

1.3. Vermessung von Wänden

Um den Moskito als Vermessungsgerät einzusetzen, muss man eine dafür markierten Kante (Parallele zur Ordinatenachse) parallel zur Wand ausrichten, den Abstand zur Wand bestimmen bzw. durch das Gerät bestimmen lassen, zwei diagonale Eckpunkte der Wand anvisieren und bestätigen. Danach ist das Gerät in der Lage mehrere horizontale, vertikale oder auch einzelne Punkte in einem bestimmen Abstand zueinander und zu den Kanten der Wand anzuzeichnen. Es ist egal ob der Moskito in der Mitte oder am Rand der Wand steht. Empfehlenswert ist eine möglichst mittige Position, da dabei die hardwarebedingten Fehler

geringer sind. Die Wand darf keine Schrägen besitzen und muss gerade sein. Diese Funktion bietet eine Erleichterung und ein schnelleres Arbeiten für jeden Hand- und Heimwerker. Somit kann man in erheblich kürzerer Zeit Bilder aufhängen, Kleiderhaken montieren ...

Die Berechnung der Markierungspunkte geht in zwei Schritten vonstatten. Zuerst werden die Koordinaten der Eckpunkte berechnet. Dazu werden die Gleichungen aus den vorhergehenden Kapiteln genutzt. Die x-Koordinate entspricht dem Abstand der Wand zur Kante plus dem Abstand der Kante zum Koordinatenursprung, da das Gerät entsprechend ausgerichtet sein muss. Mithilfe der gemessenen Winkel kann man die restlichen Koordinaten bestimmen:

$$y = \frac{r - x \cdot \cos\alpha}{\sin\alpha}; (0\,^\circ < \alpha < 180\,^\circ) \tag{1.9}$$

$$z = \tan\beta \cdot \sqrt{x^2 + y^2} + h; (z > h) \tag{1.10}$$

$$z = -\frac{\sqrt{x^2 + y^2}}{\tan(180 - \beta)} + h; (z < h) \tag{1.11}$$

Die jeweiligen z- und y-Koordinaten der Eckpunkte geben die Lage der Kanten der Wand an. Gibt am dem Programm die Aufgabe einen Punkt in einem bestimmten Abstand von rechts oder links bzw. von oben oder unten anzuzeigen, so werden die Koordinaten des Punktes durch die bekannten Eckpunkte errechnet und anschließend mit den Gleichungen aus den vorhergehenden Gliederungspunkten die Drehwinkel berechnet und angefahren.

1.4. Abweichungen und Größtfehlerberechnung

Hardwarebedingt gibt es immer eine bestimmte Ungenauigkeit. Sie setzt sich zusammen aus der Genauigkeit der Servomotoren, der Größe des Lichtpunktes und dem Abstand zwischen Gerät und Ziel (bzw. Wand). Eine Winkelabweichung der verwendeten Servomotoren um $\pm 1\,^\circ$ ist in empirischen Versuchen ermittelt wurden.

Für die Höhenabweichung stellt man die Gleichung 1.7 bzw. 1.8 nach z um.

$$\Delta z = -\frac{\sqrt{x^2 + y^2}}{\tan(180 - \beta \pm 1)} + h; (z < h) \tag{1.12}$$

$$\Delta z = \frac{\sqrt{x^2 + y^2}}{\tan(\beta \pm 1)} + h; (z > h) \tag{1.13}$$

Bei $z = h$ müssen beide Formel verwendet werden. Generell sollte man prüfen, ob das errechnete z die Bedingung der Formel einhält und gegebenen Falls beide Formeln nutzen.

Die horizontale Abweichung in der x-Ebene des (Ziel-)Punktes ergibt sich aus dem Umstellen der Gleichung 1.4 nach y

$$\Delta y = \frac{r - x \cdot \cos(\alpha \pm 1)}{\sin(\alpha \pm 1)} \tag{1.14}$$

Das Programm errechnet aus diesen Formeln immer die aktuelle Ungenauigkeit.

2. Technische Umsetzung und Softwarearchitektur

Das Gerät umfasst einen Arduino Mega 2560 mit LCD-Display, Ethernet-Shield, Relais, einen Laser, zwei Servomotoren, einem Ultraschallsensor sowie viele kleine elektrische Steuerelemente (siehe Bild auf dem Deckblatt). Die Kommunikation mit der Steuersoftware des Computers erfolgt über eine USB-Schnittstelle. Zusätzlich kann das Gerät, wenn es mit einem Netzwerk verbunden ist, über dieses eine Monitorring-Webseite anzeigen lassen.

Der Quellcode der gesamten Software, das heißt sowohl das Mikrocontrollersystem, als auch das Anwenderprogramm für den Computer, findet man auf GitHub.com. Der genaue Link befindet sich im Anhang. Bei Weiterentwicklung werden die Änderungen immer auf der entsprechenden Projektseite hochgeladen.

2.1. Herstellung und Fertigung der Bauteile

Einige Teile der Konstruktion sind aus Holz gefertigt. Diese Bauelemente haben vor allem statische Aufgaben und geben dem Gerät Aussehen und Form. Holz ist ein idealer Rohstoff, da er preiswert und umweltfreundlich in der Beschaffung und Entsorgung ist. Außerdem bietet er eine hohe Stabilität bei geringem Gewicht. Zur Fertigung der Bauelemente diente ein 3D-Verfahren mithilfe einer CNC-Maschnie. Alle entstandenen Teile mussten in mehreren Schritten gefertigt werden und bestehen aus mehreren Komponenten, die durch Fügeverfahren miteinander verbunden wurden. An besonders wichtigen Stellen wurden mehrere Fügeverfahren in Kombination verwendet. Die Konstruktionsdateien der Fräsmaschine befinden sich ebenfalls auf GitHub.

Neben den Holzbauten nutzt das Gerät auch eigene Schaltkreise, die mit dem Arduino verbunden sind und über diesen mit elektrischen Strom versorgt werden. So werden die oben genannten elektrischen Bauelemente über ein Breadboard korrekt und platzsparend angeschlossen. Der Schaltkreis ist in der Lage größere Netzschwankungen, die zum Beispiel durch das Anschalten des Lasers entstehen, abzufangen und alle elektrischen Bauelemente immer mit dem nötigen Strom zu versorgen. Wäre dies nicht der Fall, so würde bei jedem Anschalten des Lasers das LCD-Display ausgehen. Diese energiesparende Methode ist nicht nur ökonomisch, sondern vor allem auch umweltschonend.

Bei der Entwicklung wurde mit keinem Institut, keiner Hochschule und keinen anderen Organisation zusammengearbeitet. Es ist somit allein das Produkt meiner Arbeit ohne fremde Hilfe. Das betrifft nicht nur den Bau, sondern auch die Programmierung und das Testen des Gerätes mit all seinen Fassetten.

2.2. Softwarearchitektur und Programmierung

2.2.1. layered system structure

Das Softwarekonzept (siehe Abb. 2.1) sieht drei verschiedene Schichten vor. Die grundlegenste Schicht ist das Mikrocontrollersystem (die blauen und gelben Kästchen im Bild), welches die Hardware (graue Kästchen) direkt steuern und auslesen kann. Es ist universal, bietet jedoch für den Benutzer wenig Zugriffsmöglichkeiten. Die Aufgabe des Systems ist

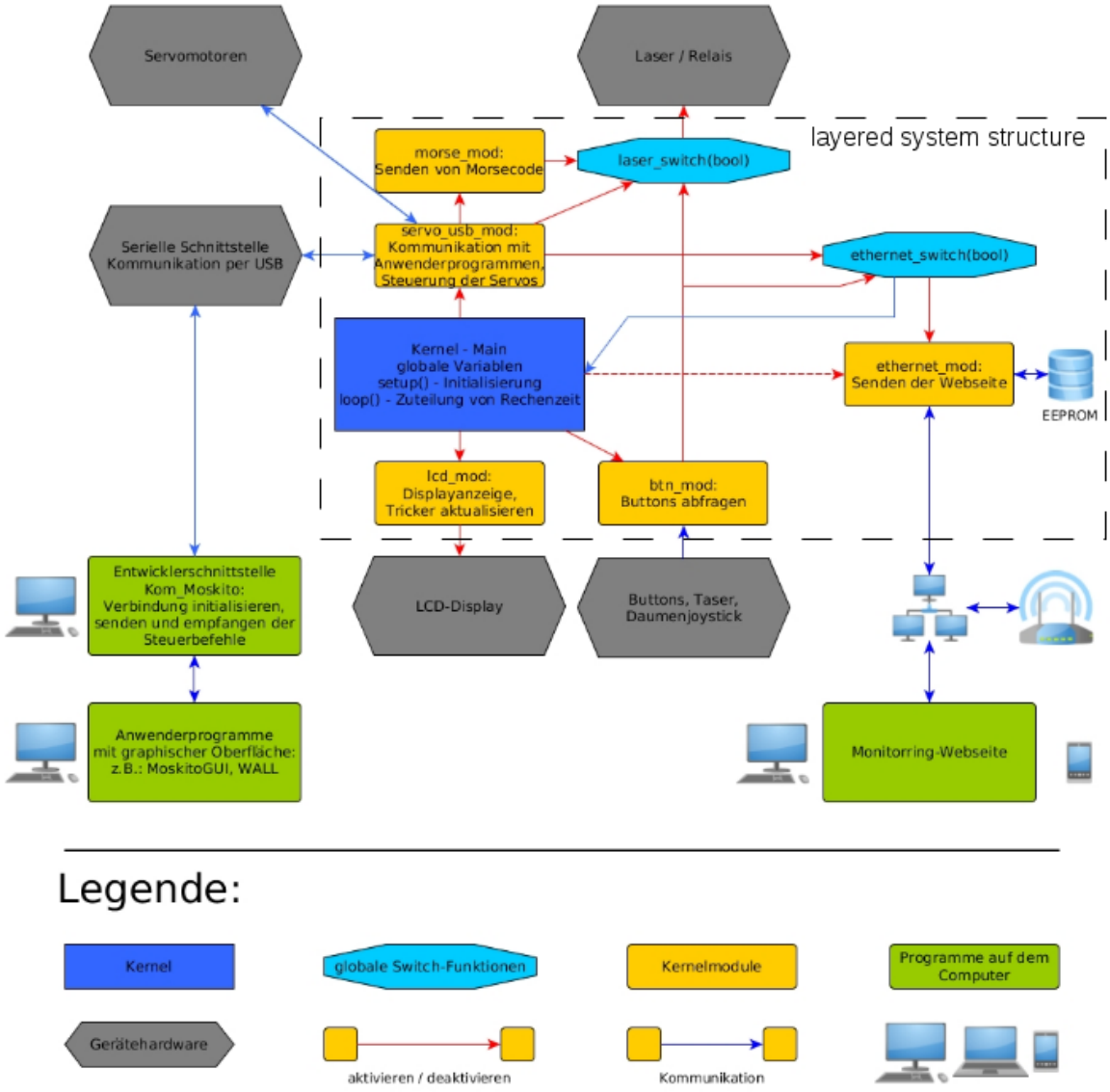


Abbildung 2.1.: Softwarearchitektur des Projektes und besonders des Mikrocontrollersystems

es das LCD-Display, die Servomotoren, das Ethernetshield, die Kommunikation mit anderen Schichten und die Direkteingabe durch Taster zu managen.

Die zweite Schicht sind Anwenderprogramme am Computer (grüne Kästchen links im Bild). Diese Programme können beliebig und je nach Aufgabe des Systems ausgetauscht werden. Durch eine grafische Oberfläche bieten sie Benutzerfreundlichkeit und Funktionalität für den bestimmten Zweck, für den sie konzipiert sind. Eine Entwicklerschnittstelle in Form einer Qt4-Klasse ermöglicht Entwicklern sicheren und exakten Zugriff auf das Gerät. Auch wenn der Entwickler keine Ahnung von der Kommunikation und den Steuerbefehlen des Moskito hat, ist er somit in der Lage ihn vollständig zu nutzen. Durch unterschiedliche Anwenderprogramme mit verschiedenen Aufgaben und Zielen erreicht man eine hohe Vielseitigkeit. Ein weiterer Vorzug ist, dass auch fremde Entwickler leicht ein Anwendungsprogramm schreiben können, vielleicht für einen Zweck, an den bis jetzt noch niemand gedacht hat.

Die dritte Ebene ist eine Monitorring-Webseite (grünes Kästchen rechts im Bild). Sie wird vom Mikrocontroller erstellt. Schließt man das Gerät über ein LAN-Kabel an ein Netzwerk an und aktiviert die Ethernetfunktion, so versucht das System sich per DHCP zu verbinden. Bei korrekter Einwahl kann man durch Aufrufen der entsprechenden IP-Adresse im Browser den Moskito auf seiner Webseite besuchen. Die Seite enthält neben einer kurzen Beschreibung des Projektes auch die aktuelle Position der Servomotoren und den Status des Lasers (Monitorring). Während die Beschreibung für jeden Besucher der Webseite lesbar ist, kann man das Monitorring nur nach einer entsprechenden Authentifizierung nutzen.

2.2.2. graphische Benutzeroberfläche

Eine graphische Benutzeroberfläche ermöglicht auch unerfahrenen Nutzern eine einfache Interaktion mit dem Gerät. Auf diesem Weg verbindet man eine einfache Handhabung mit einer Fülle von Funktionen für einen expliziten Anwendungsbereich. Man erreicht Multifunktionalität durch unterschiedliche Anwenderprogramme. Über die speziell entwickelte Entwicklerschnittstelle können komplexe Features in eine Reihe von Steuerbefehlen für den Moskito umgesetzt werden. Die Vielfalt der Anwenderprogramme ist dabei lediglich durch die Kreativität und die Ziele des Entwicklers limitiert. Auch ein fremder Entwickler, der keinerlei Ahnung vom Moskito besitzt kann durch die Entwicklerschnittstelle einfach und korrekt das Gerät steuern.

Diese Schnittstelle ist eine Qt-Klasse namens 'Kom_Moskito' (in der Datei 'lib/Kom_Moskito.h'). Qt bietet sich als Grundlage der grafischen Oberfläche an, da das gesamte Projekt durchgehend in C++ geschrieben ist und Qt grafische Benutzerinteraktionen bereitstellt. Um sie zu benutzen legt man ein Objekt der Klasse 'Kom_Moskito' an. Mit der Methode 'bool init(QString path)' baut man die Verbindung zum Gerät auf. Anschließend kann man durch Verwendung der restlichen Methoden Abfragen oder Steuerbefehle auslösen. Das Listing 2.1 zeigt einen regulären Zugriff auf das Gerät vom Computer aus.

Listing 2.1: Die Verwendung der Entwicklerschnittstelle Kom_Moskito

```
#include "[...]/lib/Kom_Moskito.h"

int main(int argc, char *argv[])
{
    Kom_Moskito *moskito = new Kom_Moskito;
    moskito->init("/dev/ttyACM0"); // Moskito initialisieren
    moskito->aim_deg(32, 105, 1);  // positionieren (Alpha=32;
                                   // Beta=105; Laser an)
    moskito->setLaser(false);      // Laser ausschalten
    // ...
}
```

Zur Zeit gibt es zwei verschiedene Anwendungen, die auf 'Kom_Moskito' aufgebaut sind.

WALL Das WALL-Programm ermöglicht es Hand- und Heimwerkern unkompliziert Wände zu vermessen und zum Beispiel Bohrpunkte zu markieren. Dazu stellt man das Gerät parallel zur Wand, fährt den Laser durch manuelle Steuerung an zwei gegenüberliegende Eckpunkte und klickt auf Eingabe. Das Programm schafft anschließend die Möglichkeit einzelne Punkte oder horizontale bzw. vertikale Linien aus mehreren Punkten zu zeichnen. Es spielt keine Rolle, ob der Moskito in der Mitte oder am Rand der Wand steht. Aus Gründen der Genauigkeit empfehle ich jedoch den Moskito möglichst in der Mitte (sowohl horizontal als auch vertikal) zu orientieren. Parallel wird vom Programm auch eine Fehlerrechnung mit Größtfehlerabschätzung des jeweiligen Punktes angezeigt.

MoskitoGUI Ziel des MoskitoGUIs (GUI steht für General User Interface) ist es eine Oberfläche anzubieten, welche das Anpeilen bestimmter Koordinaten im Raum ermöglicht. Dazu gehören umfangreiche Speicher- und Verwaltungsmöglichkeiten wichtiger Koordinaten. So kann man markante Punkte speichern um sie jederzeit blitzschnell wieder anzupeilen.

2.2.3. Monitorring-Webseite

Aktiviert man das Ethernet (durch einen Button oder serielle Steuerbefehle), so versucht das Gerät sich mithilfe von DHCP oder einer statischen IP einzuwählen. Bei erfolgreicher Einwahl wird die IP-Adresse des Moskito im Tricker des LCD-Displays angezeigt. Gibt man in einen Browser diese IP-Adresse ein, so sieht man eine Weboberfläche mit mehreren Seiten. Wichtig zu beachten ist, dass der Computer sich im selben Netzwerk wie der Moskito befinden muss, damit eine Verbindung aufgebaut werden kann.

Die Webseite umfasst eine kurze Erklärung des Projektes und zeigt in einem passwortgeschütztem Bereich die aktuelle Position bzw. den Laserstatus an.

Passwortverwaltung und permanente Speicherung mit EEPROM

Der Nutzername und das Passwort für den geschützten Bereich ist verschlüsselt im permanenten Speicher des Mikrocontrollers (EEPROM) hinterlegt. Die Zugangsdaten lassen sich über das Protokoll der seriellen Kommunikation ändern.

2.2.4. modulbasiertes Softwaredesign des Mikrocontrollersystems

Wie man in der Abbildung 2.1 sieht, besteht das Grundsystem des Mikrocontrollers aus mehreren Modulen. Ziel ist es immer wiederkehrende Aufgaben stets von den selben Funktionen und Modulen bearbeiten zu lassen. Die einzelnen Module sind somit für ihre Aufgabe entsprechend spezialisiert. Das Auftreten der Fehler ist durch Qualifizierung der Software stark reduziert. Somit werden zum Beispiel bestimmte Hardwarekomponenten nur von bestimmten Modulen angesprochen und verwaltet. Zusätzlich ermöglicht die modulare Programmierung in diesem Fall eine gute Performance, da die Module nur ausgeführt werden, wenn dies notwendig ist.

Kernel

Der Kernel ist der Teil des Codes, der sich um den Startvorgang und die Vergabe von CPU-Zeit kümmert. Im Kernelbereich des Quelltextes werden zusätzlich alle nötigen Bibliotheken hinzugefügt, sowie globale Variablen und Funktionen definiert.

CPU-Allocation in der Kernelfunktion loop() Die Programmierung muss mehrere Aufgaben nahezu gleichzeitig bearbeiten. Um dies zu bewältigen gibt es für jedes Modul eine runtime-Funktion, die die Anfragen und Aufgaben des entsprechenden Moduls bearbeitet. Diese Funktionen müssen stets in wenigen Millisekunden abgearbeitet sein, damit man den Eindruck des Multithreading erhält. Die runtime-Funktionen werden entsprechend oft oder weniger oft aufgerufen und somit Rechenzeit zugeordnet.

Das wichtigste Modul heißt 'servo_usb_mod'. Es ermöglicht die Kommunikation per USB und die Steuerung der Servomotoren. Dieses Modul ermöglicht es Anwenderprogramme zu nutzen. Es bekommt $41,6\%$ Rechenzeit bei eingeschaltetem Ethernet und 50% bei ausgeschaltetem Ethernet.

Zusätzlich befinden sich in manchen Funktionen Prozeduren, die nur nach einer bestimmten Zeit ausgeführt werden, egal wie oft in diesem Zeitraum die Funktion durchlaufen wird. Dies ist zum Beispiel beim Weiterrücken des Trickers auf dem LCD-Display oder beim Abtasten des Buttons der Fall.

Listing 2.2: Das Modul: btn_mod

```
/*************************************************************
       Button - Modul
*************************************************************/
void btn_runtime()
{
  if ((millis() - btn_t) >= BTN_TIME){ // Zeitsteuerung
    if (digitalRead(BUTTON_A_PIN)){ laser_switch(!LASER);}
    if (digitalRead(BUTTON_B_PIN)){ ethernet_switch(!ethernet);}
    // ...
    btn_t = millis();
  }
}
```

Kommunikation per USB (servo_usb_mod)

Die Datenübertragung verläuft über ein speziell dafür entwickeltes Protokoll. Dieses Protokoll ist so konzipiert, dass es möglichst schnell wenige Steuerbefehle empfangen und Abfragen beantworten kann. Jeder Steuerbefehl hat ein bestimmtes eineindeutiges Zeichen inne. Dieses Zeichen wird am Anfang eines Steuerkommandos gesendet. Danach kommen mögliche Parameter, die jeweils durch '#' voneinander getrennt sind. Am Ende eines Befehls mit Parametern kommt erneut '#' um den Befehl zu beenden. Bei Befehlen, die keine Parameter besitzen sendet man lediglich das Zeichen.

Der Befehl zum Positionieren des Servomotors heißt:

Listing 2.3: Steuerbefehl a (positioniert die Servomotoren)

```
Syntax  :   a<alpha>#<beta>#<laser>
Beispiel:   a34#67#1
```

Der erste Parameter gibt die Position des Alphaservos ($\alpha = 34\,°$) und der zweite die Position des Betaservos ($\beta = 67\,°$) an. Der letzte Parameter gibt den Status des Lasers an. 1 bedeutet der Laser wird aktiviert, 0 bedeutet deaktiviert und $*$ heißt der Laser bleibt unverändert. Grundsätzlich wird während der Drehung der Servomotoren der Laser aus Sicherheitsgründen abgeschaltet und gegebenen Falls danach wieder eingeschaltet.

Die Steuerbefehle geben jedoch keine Antwort. Um dennoch die aktuelle Position und die Zustände des Lasers und des Ethernets zu erfahren gibt es den Befehl ' i '. ' i ' wird nicht zu den Steuerbefehlen gezählt, sondern bildet seine eigene Kategorie der Abfragen. Die Ausgabe des Befehls ist ein String, bei dem die einzelnen Parameter durch '#' voneinander getrennt werden.

Listing 2.4: Abfrage i

```
Syntax der Ausgabe : <alpha>#<beta>#<laser>#<ethernet>
Beispiel           : 34#67#1#0
```

Die vollständige Interpretation der einzelnen Steuerbefehle ist in der Datei 'Arduino/Moskito/servo_usb_mod.h' implementiert.

Es ist aber absolut nicht nötig diese Befehle zu kennen. Um trotzdem (auch als Entwickler) den Moskito voll und ganz zu nutzen, gibt es eine Entwicklerschnittstelle. Diese bietet eine Qt4-Klasse mit Funktionen für alle Belange. Sie ist in der Datei 'lib/Kom_Moskito.h' implementiert.

Berücksichtigung der Wiederholgenauigkeit der Servomotoren in der Programmierung

Vor allem bei Positionsänderungen um wenige Grad haben die Servomotoren große Ungenauigkeiten. So kann es passieren dass der Servomotor nicht weiter rückt und beim nächsten mal gleich zwei Grad weiterfährt. Um dem entgegen zu wirken werden geringfügige Positionsänderung durch spezielle Funktionen bearbeitet. So fährt der Servomotor erst $10\,^\circ$ weiter und danach $9\,^\circ$ zurück. Steht der Servomotor zum Beispiel bei $178\,^\circ$ und soll ein Grad weiter rücken, so wird zuerst $10\,^\circ$ zurück gefahren und $11\,^\circ$ nach vorn.

Diese Korrekturmaßnahme steigert die Wiederholgenauigkeit enorm. Bei einem Experiment konnte die Genauigkeit um bis zu 80% erhöht werden. Bei größeren Positionsänderungen ist die Anfahrgenauigkeit hoch und es müssen keine Korrekturen vorgenommen werden.

LCD-Modul (lcd_mod)

Das LCD-Modul steuert das Display des Mikrocontrollers. Es bildet die aktuellen Servomotorpositionen ab und zeigt einen Tricker. Über diesen Tricker kann man sich bestimmte Texte und den Ethernetstatus anzeigen lassen. Wie man im Listing 2.5 sieht, besitzt die runtime-Funktion zwei Zeitschaltuhren. Das heißt bestimmte Funktionen wie 'lcdRefresh()' oder der Tricker werden erst nach einer bestimmten Zeit wieder ausgeführt, egal wie oft die Funktion in diesem Zeitraum durchlaufen wurde. So rückt zum Beispiel der Tricker nur nach 'TRICKER_TIME' Millisekunden (nach Standardeinstellungen $0,8s$) auf die nächste Stelle weiter. Der gesamte Bildschirm wird nach acht Sekunden komplett gelöscht und wieder neu geschrieben um mögliche Darstellungsfehler zu korrigieren.

Listing 2.5: runtime-Funktion des LCD-Moduls

```
void lcd_runtime()
{
  if ((millis() - clear_t) >= CLEAR_TIME){    // Bildschirm reseten
    lcdRefresh();
    clear_t = millis();
  }

  if ((millis()- tricker_t) >= TRICKER_TIME){ // Tricker akutalisieren
    if ((tricker_all.length()) > LCD_LAENGE){
      writeTricker();
      if (tricker_n >= tricker_all.length()) {tricker_n = 0;}
      else {tricker_n++;}
    }else{
      lcd.setCursor(0, 0);
      lcd.print(tricker_all);
    }
    tricker_t = millis();
```

```
    }

    /* Position der Servomotoren in Echtzeit */
    lcd.setCursor(7, 3);
    lcd.print(alphaServo.read());
    lcd.print("  ");
    lcd.setCursor(17, 3);
    lcd.print(betaServo.read());
    lcd.print("  ");
  }
```

2.2.5. Ethernet-Modul (ethernet_mod)

Für den Aufbau und die Verwaltung einer Webseite nutzt das Ethernet-Modul (ethernet_mod) zusätzlich eine Bibliothek namens Webduino. Ein Link zur Projektseite befindet sich im Anhang. Der verwendete Code ist unter der MIT-Lizenz verfügbar und somit uneingeschränkt für das Moskitoprojekt nutzbar.

Das Netzwerkmodul unterscheidet sich von allen anderen Modulen in sofern, dass es keine Funktion namens 'ethernet_runtime()' besitzt. Stattdessen übernimmt die Methode 'WebServer::processConnection()' aus der Webduino-Bibliothek dessen Aufgabe. Die Funktionen 'adminCmd()' und 'aboutCmd()' bewerkstelligen das Senden der Seite '/admin.html' bzw '/about.html'. Der Html-Code ist im Programmspeicher gespeichert bzw. wird bei Positionsabfragen zur Laufzeit erstellt. Die Seite '/admin.html' ist ein geschützter Bereich. Um Zugang zu ihm zu haben, muss man einen existierenden Nutzernamen und ein dazugehöriges, korrektes Passwort angeben. Die Authentifizierung erfolgt über das HTTP-Protokoll standardmäßig. Auf der Adminseite sind die aktuellen Einstellungen der Servomotoren und des Lasers vermerkt.

Die globale Funktion 'ethernet_switch(bool)' aktiviert und deaktiviert das Modul. Nach dem Starten ist das Modul standardmäßig deaktiviert. Der Button 'B' und die seriellen Steuerbefehle 'e' (aktivieren) und 'o' (deaktivieren) nutzen die Switchfunktion um den Ethernetstatus zu ändern.

2.2.6. Qualitätssicherung der Softwarearchitektur

Um eine funktionstüchtige Software zu garnieren muss diese intensiv und regelmäßig getestet werden. Dabei reicht es nicht aus per Hand verschiedene Befehle einzugeben und zu kontrollieren, dass die gewünschte Aktion daraus hervorgehen.

Für den Moskito benutze ich deshalb eine Reihe von Testprogrammen, die verschiedene Abläufe mit unterschiedlichen, sich ändernden Parameter mehrfach erproben. So gibt es das Programm 'stresstest', welches den Moskito mit ständigen Positionsänderungen und Abfragen testet. Zusätzlich gibt es das Programm 'test_kom_moskito', welches alle Funktionen der Entwicklerschnittstelle mehrfach und intensiv testet. Beide Programme erstellen ein Testprotokoll. Einen Ausschnitt aus den Protokollen befindet sich im Anhang.

A. Anhang

Der gesamte Quellcode des Projektes befindet sich auch GitHub.com (`https://github.com/KonradMerkel/Moskito`). Zukünftige Änderungen am Code sowie neu hinzugekommende Programme werden natürlich auch auf GitHub veröffentlicht. Bitte scheuen Sie es nicht sich den Quellcode durchzulesen und nachzuvollziehen. Er ist gut kommentiert und logisch aufgebaut. In einer 'README.md' findet man eine kurze Erklärung in welchem Ordner welche Programme liegen und womit man anfangen sollte.

Im folgenden Literaturverzeichnis sind alle zusätzlichen Bibliotheken aufgeführt, die in das Projekt eingebunden wurden. Diese stehen alle unter der MIT-Lizenz und sind somit für das Projekt ohne weiteres nutzbar.

A.1. Auszüge aus den Testprotokollen

A.1.1. Stresstest

Ziel des Stresstestes ist es, so viel wie möglich Positionsänderungen durchzuführen und zu kontrollieren, ob die Position erfolgreich angefahren ist. Dabei wird vor allem die Kommunikation zwischen Moskito und Computer getestet.

Listing A.1: Auszug aus dem Stresstest

```
$./stresstest
1: [OK] Alpha 33 == 33
2: [OK] Beta 67 == 67
3: [OK] Alpha 28 == 28
4: [OK] Beta 11 == 11
5: [OK] Alpha 117 == 117
6: [OK] Beta 119 == 119
7: [OK] Alpha 138 == 138
8: [OK] Beta 23 == 23
9: [OK] Alpha 11 == 11
10: [OK] Beta 12 == 12
11: [OK] Alpha 86 == 86
12: [OK] Beta 147 == 147
13: [OK] Alpha 9 == 9
14: [OK] Beta 128 == 128
15: [OK] Alpha 51 == 51
16: [OK] Beta 79 == 79
17: [OK] Alpha 101 == 101
18: [OK] Beta 43 == 43
19: [OK] Alpha 30 == 30
20: [OK] Beta 161 == 161
...
284: [OK] Beta 144 == 144
285: [OK] Alpha 178 == 178
286: [OK] Beta 164 == 164
287: [OK] Alpha 107 == 107
288: [OK] Beta 143 == 143
289: [OK] Alpha 122 == 122
290: [OK] Beta 177 == 177
291: [OK] Alpha 153 == 153
292: [OK] Beta 122 == 122
293: [OK] Alpha 138 == 138
```

```
294: [OK] Beta 161 == 161
295: [OK] Alpha 144 == 144
296: [OK] Beta 104 == 104
297: [OK] Alpha 104 == 104
298: [OK] Beta 128 == 128
299: [OK] Alpha 145 == 145
300: [OK] Beta 156 == 156

[OK]  {No error.}
[OK]  {No warning.}
[OK]  {No bugs.}
```

A.1.2. test_kom_moskito

Mit dem Test 'test_kom_moskito' wird jede Funktion in der Entwicklerschnittstelle aufgerufen und die Reaktion überprüft. Die Parameter für die Eingabe variieren um auch nicht offensichtliche Fehler zu finden.

Listing A.2: Auszug aus dem test_kom_moskito-Test

```
$ ./test_kom_moskito
1: [OK] Initialisierung(PORT) 1 == 1
2: [OK] setLaser(bool) (only return) 1 == 1
3: [OK] getLaser() (+ setLaser(bool)) 0 == 0
4: [OK] setLaser(bool) (only return) 1 == 1
5: [OK] getLaser() (+ setLaser(bool)) 1 == 1
6: [OK] setLaser(bool) (only return) 1 == 1
7: [OK] getLaser() (+ setLaser(bool)) 1 == 1
8: [OK] setLaser(bool) (only return) 1 == 1
9: [OK] getLaser() (+ setLaser(bool)) 0 == 0
10: [OK] standby() (only return) 1 == 1
11: [OK] pos(int, int, bool, bool) (only return) 1 == 1
12: [OK] alpha: pos(int, int, bool, bool) 90 == 90
13: [OK] beta: pos(int, int, bool, bool) 90 == 90
14: [OK] laser: pos(int, int, bool, bool) 0 == 0
15: [OK] ethernet: pos(int, int, bool, bool) 0 == 0
16: [OK] send(QString) (only return) 1 == 1
...
68: [OK] setEthernet(bool) (only return) 1 == 1
69: [OK] getEthernet() (+ setEthernet(bool)) 1 == 1
70: [OK] setEthernet(bool) (only return) 1 == 1
71: [OK] getEthernet() (+ setEthernet(bool)) 0 == 0
72: [OK] Initialisierung(Testmodus) 1 == 1

[OK]  {No error.}
[OK]  {No warning.}
[OK]  {No bugs.}
```

A.2. Importierte Bibliotheken

- Qextserialport: Eine Qt-Erweiterung zur Unterstützung von seriellen Schnittstellen `http://code.google.com/p/qextserialport/`
- Webduino: Eine Bibliothek um den Arduino als Webserver zu nutzen `https://github.com/sirleech/Webduino`