

Master/Slave Modul für den CTBot

Ausarbeitung zum Wahlpflichtfach Mobile Roboter

Matthias Geller (26750)

21.07.09

Inhaltsverzeichnis

| | |
|---|----|
| Einleitung..... | 3 |
| Themenbeschreibung..... | 3 |
| Wissenswertes vorab..... | 3 |
| Voraussetzungen..... | 3 |
| Die Entwicklungsumgebung..... | 4 |
| Das VMware Image..... | 4 |
| Allgemein..... | 4 |
| Die Struktur..... | 4 |
| Die installierte Software..... | 4 |
| Putty..... | 4 |
| Eclipse..... | 5 |
| AVR Studio4..... | 5 |
| Die Basiskonfiguration der Roboter..... | 6 |
| Serielle Konfiguration mittels Putty..... | 6 |
| Schritt 1: Bot und PC verbinden..... | 6 |
| Schritt 2: PC Hardware Manager überprüfen..... | 7 |
| Schritt 3: Das Putty Konfigurationsfenster..... | 7 |
| Schritt 4: Mit x ins Konfigurationsmenu..... | 8 |
| Schritt 5: Die einzelnen Konfigurationseinstellungen..... | 8 |
| 0 Server..... | 8 |
| 1 Channel 1..... | 8 |
| 2 Channel 2..... | 9 |
| 3 E-mail..... | 9 |
| 4 WLAN..... | 9 |
| 5 Expert..... | 9 |
| 6 Security..... | 10 |
| 7 Defaults..... | 10 |
| 8 Exit without save..... | 10 |
| 9 Save and exit..... | 10 |
| Schritt 6: Das Webinterface testen..... | 10 |
| Das Master/Slave Modul..... | 11 |
| Was bedeutet Master/Slave?..... | 11 |
| Warum heisst es Master/Slave Modul?..... | 11 |
| Kommandos und Subkommandos – SIGNALS und SUBSIGNALS..... | 11 |
| Die Funktionsweise des Master/Slave Moduls..... | 12 |
| Funktionsbeschreibung..... | 13 |
| Abschluss..... | 15 |
| Literatur..... | 15 |
| Hinweis..... | 15 |
| Quellen..... | 15 |

Einleitung

Themenbeschreibung

Diese Ausarbeitung befasst sich mit der Implementierung eines Master/Slave Moduls für den CTBot, welches an die Arbeit von C. Seitzer, M. Bellgardt, H- Maier und M. Barth mit dem Namen „Mobile Roboter – Dokumentation für Mobile Roboter im Hauptstudium“ anknüpfen soll. Für die Demonstration einer praktischen Anwendung wurde eine verteilte Suche nach einer Lichtquelle implementiert. Um zukünftigen Studenten eine einsatzbereite Ausgangsplattform für eine möglichst unkomplizierte Weiterentwicklung des CTBot Projekts zu bieten wird in diesem Projekt auch ein VMware Image mit für diesen Zweck benötigten Komponenten erstellt.

Wissenswertes vorab

Das CTBot Projekt der Hochschule ist eine ausgezeichnete Möglichkeit, sich mit den Funktionsweisen eines kleinen Roboters vertraut zu machen. Es sei aber davor gewarnt zu glauben, dass man sich ausschliesslich mit der Programmierung befassen kann ohne die Tücken der Hardware zu kennen. Leider ist der CTBot etwas anfällig auf fehlerhafte Hardwarekomponenten, kalte Lötstellen und hat zudem manches mal Probleme mit ungenauen bzw. defekten Sensoren. Dies sollte aber keinen interessierten Studenten davon abbringen lassen, sich diesem Thema anzunehmen. Zur Implementierung des Master/Slave Moduls lässt sich vorab noch sagen, dass es noch viele Optimierungsmöglichkeiten bietet, welche ich erst teilweise nach Abgabe dieser Arbeit angehen werde.

Voraussetzungen

Für die Modifikation und Weiterentwicklung des Projekts sind der Umgang mit Entwicklungswerkzeugen wie z.B. Eclipse und ausreichend Erfahrung in der C/C++ Programmierung von Vorteil. Das VMware Image ist in seiner Grösse auf 8GB beschränkt. Rechnet man nun eventuelle Snapshots hinzu, so kann der Ordner mit den VMware Dateien schnell auf 9GB anwachsen, wenn man die Snapshots selbst noch sichern möchte. Es sollte daher darauf geachtet werden, dass ausreichend Plattenspeicher auf dem eingesetzten Computer zur Verfügung steht. Falls man den CTSim weiterentwickeln bzw. nutzen möchte, braucht man hierfür einen geeigneten VMware Player welcher DirectX Unterstützung bietet. Leider ist in den kostenlosen Versionen von VMware Playern, welche ich im Netz gefunden habe, die DirectX Unterstützung deaktiviert. Diese VMware wurde mit VMware Fusion (Version 1.1.3) erzeugt und mit VMware Workstation (Version 6.0.0) getestet. Falls man aller Voraussicht nach keinen Simulator benötigen wird, ist die DirectX Unterstützung sekundär und nicht erforderlich.

Die Entwicklungsumgebung

Das VMware Image

Allgemein

Eine sofort einsatzbereite, einheitliche sowie plattformunabhängige Entwicklungsumgebung zu schaffen begründete das Ziel, ein VMware Image speziell für die Bedürfnisse der CTBot Programmierung zu erstellen. Im Folgenden werden hier die einzelnen Bestandteile des VMware Images erläutert, welche für ein sinnvolles Arbeiten relevant sind. Es sei an dieser Stelle darauf hingewiesen, dass diese erste Version der VMware nur grundlegende Funktionalität bereitstellt, welche auf dem Stand von heute beruhen. Das CTBot Projekt versteht sich aus meiner Sicht heraus als dynamisches Projekt, welches von zukünftigen Studenten weiterentwickelt werden soll und mit einer einheitlichen Entwicklungsumgebung dies auch kann. Als ein eventuell interessantes Beispiel für die Erweiterung der Funktionalität könnte die Idee eines über das Internet zugängliches Subversion Repository sein. Auf überflüssige Funktionen sollte in diesem Zusammenhang aber weitestgehend verzichtet werden, um die Übersichtlichkeit nicht zu verlieren und das auf 8GB beschränkte VMware Image nicht zu sprengen.

Die Struktur

Die Struktur ist möglichst einfach gehalten. Es gibt im Laufwerk C einen Ordner CTBot, welcher die beiden Ordner workspace und backup enthält. Der Ordner backup ist selbsterklärend. Der Ordner workspace wird von Eclipse verwendet und enthält momentan drei Projektverzeichnisse, welche in Eclipse bereits eingebunden sind. In diesem Verzeichnis findet man unter anderem den offiziellen Quellcode des Bots (ct-Bot) sowie des Simulators (ct-Sim) von Heise, welcher als Subversion direkt in das workspace Directory importiert wurde. Durch einen Klick in Eclipse auf „Replace with->Latest from Repository“ im Kontextmenu des jeweiligen Verzeichnisses bzw. einer jeweiligen Datei kann man den Quellcode wieder auf den letzten offiziellen (lauffähigen) Stand bringen. Dies ist zum Beispiel dann hilfreich, wenn man Codesegmente austesten will und im späteren Verlauf den Ausgangszustand wieder herstellen möchte. Mit einem schwarzen Stern gekennzeichnete Dateien bzw. Ordner deuten darauf hin, dass diese vom Original abweichen. Projekte welche einem Subversion Repository angehören werden zusätzlich mit (svn) im Projektnamen klar hervorgehoben. Das Projekt ct-HTW kennzeichnet die CTBot Arbeitsmappe, welches von Studenten fortgesetzt werden kann. Etwaige Milestones sollten grundsätzlich in gezippter Form im Verzeichnis backup/milestones abgelegt werden. Zum packen von Dateien eignet sich 7-Zip, welcher dafür installiert wurde. Dies gilt natürlich nur solange, bis ein entsprechendes Subversion Repository erstellt wird. Der Ordner documents dient zur Sammlung von themenspezifischen Artikeln. In ihm findet man auch die erste, nicht kontrollierte Fassung dieser Ausarbeitung, sowie nützliche Links zu Webseiten.

Die installierte Software

Putty


Mit dem Freeware Telnet/SSH Client Programm Putty kann man eine serielle Verbindung zum CTBot herstellen. Die Oberfläche von Putty ist mit ein wenig Grundverständnis intuitiv zu bedienen. Im Kapitel Die Basiskonfiguration der Roboter findet man im Abschnitt Serielle Konfiguration mittels Putty eine ausführliche und bebilderte Anleitung, wie man das Programm für unsere Zwecke zu bedienen hat. Für weiterführende Informationen lohnt ein Blick auf den offiziellen Webauftritt, welcher unter <http://www.chiark.greenend.org.uk/~sgtatham/putty/> zu

erreichen ist.

Eclipse

Die modulare Programmierumgebung Eclipse ist auf die Bedürfnisse für die Programmierung des CTBot in C/C++ aber auch für die Weiterentwicklung des Simulators CTSim mittels Java vollständig eingerichtet. Die frisch erschienene Version mit dem Namen Galileo besitzt die Build id 20090619-0625. Zum heutigen Zeitpunkt waren einige Module für Galileo (z.B. Eclox/Doxygen) nicht verfügbar und sollten bei Bedarf nachinstalliert werden. Die Installationsquelle für das Doxygen PlugIn wurde bereits zu den bestehenden Quellen hinzugefügt. Beim Kompilieren kommt es zu einer Warnung, welche „Invalid project path: Duplicate path entries“ in der Fehlerausgabe ausgibt. Für Bug ist das CDT PlugIn verantwortlich und unter der Nummer 206372 seit langem bekannt. Der Bug tritt auf, wenn man Verzeichnisse bzw. Dateien in seinem Projekt umbenennt und kann getrost ignoriert werden. Weiterführende Informationen können unter https://bugs.eclipse.org/bugs/show_bug.cgi?id=206372 nachgelesen werden. Auf die Bedienung, sowie Features von Eclipse wird in diesem Dokument nicht weiter eingegangen. Bei einer Suche nach „Tutorial Eclipse“ lieferte Google 5.680.000 Ergebnisse. Ein weiteres interessantes Feature welches in dieser ersten Version der VMware noch nicht implementiert wurde, ist die Möglichkeit in Eclipse Externe Tools einzubinden. Ein externes Tool könnte hier dann z.B. eine .bat Datei sein, welche das .hex File automatisch an den Bot überträgt, oder externe Tools welche das Debuggen unterstützen. Momentan übernimmt diese Aufgabe das AVR Studio4.

AVR Studio4

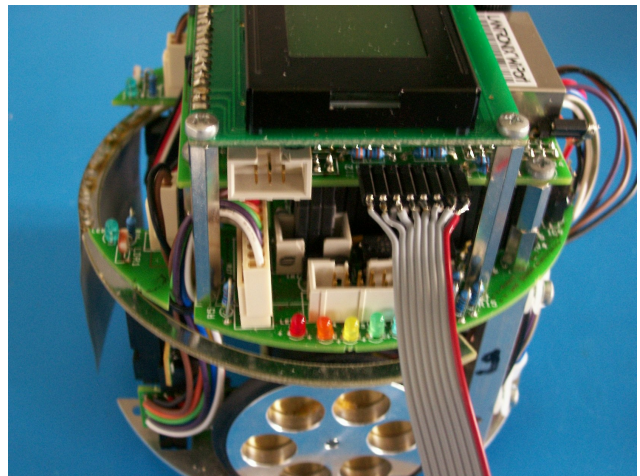
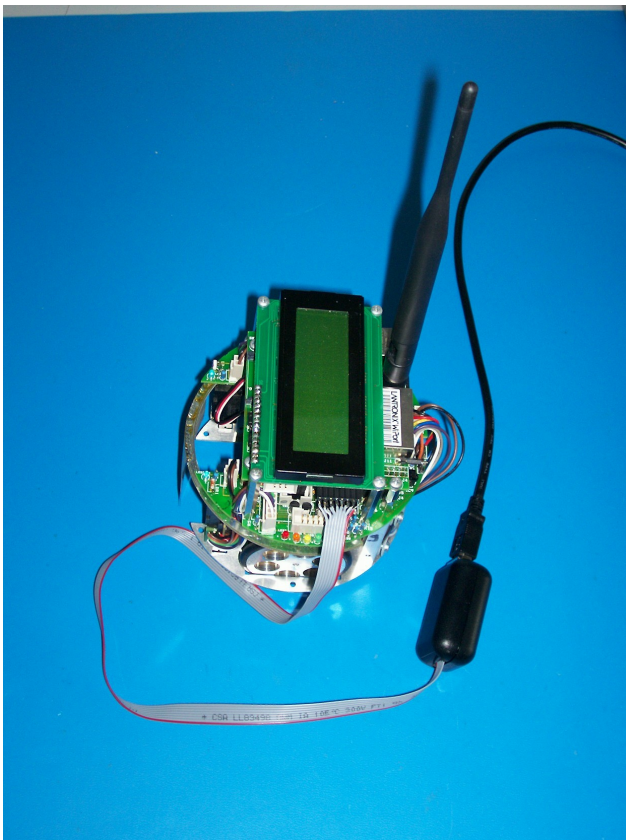
Das AVR Studio4 von Atmel ist ähnlich wie Eclipse eine vollständige IDE. Interessant hierbei ist die integrierte GUI zum Setzen bzw. Auslesen von Fuse Bits bzw. dem Flashen von .hex Files, welche unter Tools->Program AVR->Auto Connect... oder das  Icon zu finden ist. Da die an der Hochschule vorhandenen Programmieradapter alle über den USB Anschluss angeschlossen werden, werden diese automatisch vom PC erkannt und eingebunden. Das oben genannte AVR Tool erkennt den jeweiligen Adapter selbstständig und korrekt und man muss sich in der Regel keine weiteren Gedanken über das Flashen der Bots machen. Eine ausführlich bebilderte Beschreibung, wie die Fuse Bits gesetzt sein müssen und wie man das Tool zu bedienen hat liegt im Labor bereit oder bekommt es auf Nachfrage. Sollte ich das Script in elektronischer Form noch vor Abgabe meiner Arbeit auftreiben können, wird man es im Ordner documents finden können.

Die Basiskonfiguration der Roboter

Serielle Konfiguration mittels Putty

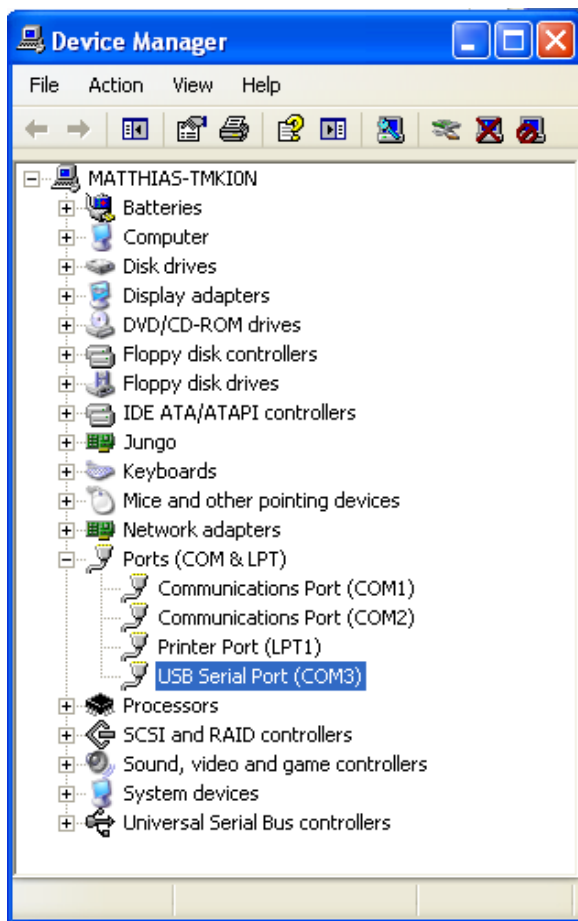
Für eine reibungslose Kommunikation ist es unerlässlich, die Grundeinstellungen seiner Bots zu überprüfen und auf einen funktionierenden Zustand zu bringen. Funktionierender Zustand soll heissen, dass diese Einstellungen mehrmals getestet wurden und in jedem Fall funktionierten und somit auch in Zukunft funktionieren werden. Ein anschliessendes Experimentieren mit den Konfigurationsmöglichkeiten des Lantronix WiPort® Moduls ist natürlich ein absolutes muss. Der Einfachheit halber wird die Konfiguration als eine Art Bilderserie aufgezeigt. Auf die wichtigsten Punkte wird explizit nochmals hingewiesen.

Schritt 1: Bot und PC verbinden



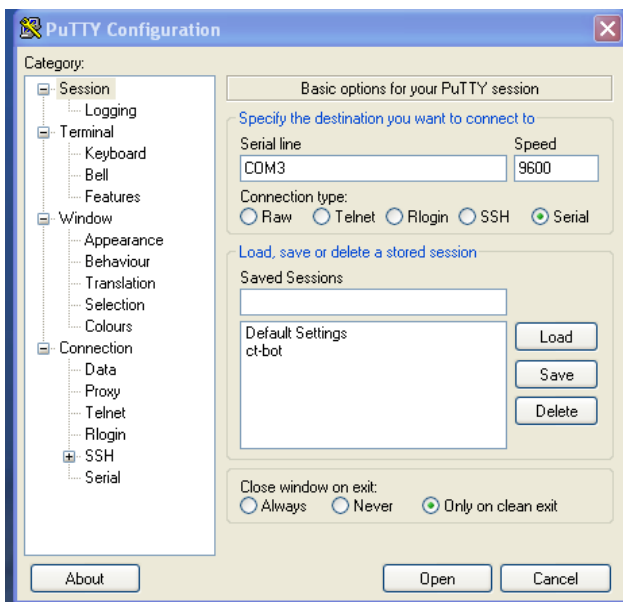
Beim Anschluss des seriellen Kabels muss darauf geachtet werden, dass die rote Litze zum WiPort® Modul hin (hier nach rechts) zeigt.

Schritt 2: PC Hardware Manager überprüfen

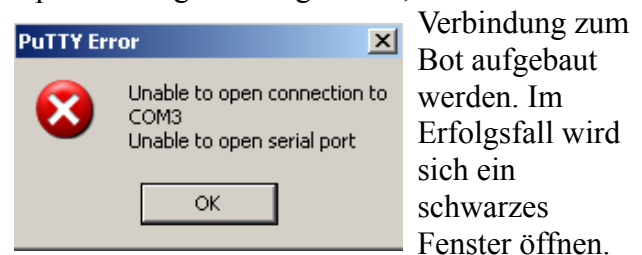


Im Hardware Manager von Windows kann überprüft werden, ob der Adapter erkannt und korrekt eingebunden wurde. Hierfür muss der Bot eingeschaltet sein. Der USB Serial Port (COM3) ist die Schnittstelle welche wir uns merken müssen. Der Port kann unter Umständen abweichen und muss dementsprechend im weiteren Verlauf jeweils entsprechend angepasst werden.

Schritt 3: Das Putty Konfigurationsfenster



Die Option Serial bei Connection type auswählen und dann den entsprechenden COM Port von Schritt 2 als Serial line verwenden. Sollte sich nach einem Klick auf den Button Open der folgende Nag öffnen, konnte keine



Achtung: COM1 und COM2 lassen immer ein schwarzes Fenster aufpoppen, sind aber in der Regel nicht die korrekten Ports für die Verbindung mit dem Bot.

Schritt 4: Mit x ins Konfigurationsmenu

Um nun in das Konfigurationsmenu zu gelangen schaltet man den Bot aus und wieder an und drückt ein paar mal die x Taste. Es sollte innerhalb von 3 Sekunden im Fenster zusammen mit der Firmwareversion und anderen Dingen die Aufforderung stehen, Enter zu drücken. Dies sollte man dann auch zügig befolgen, da diese Option auch an einen 5 Sekunden Timeout gebunden ist.

```
Change Setup:
0 Server
1 Channel 1
2 Channel 2
3 E-mail
4 WLAN
5 Expert
6 Security
7 Defaults
8 Exit without save
9 Save and exit      Your choice ?
```

Es gibt neben dem Konfigurationsmenu (x) noch sogenannte Monitormodes welche mit y bzw. z aufgerufen werden können. Der einzige Unterschied der beiden Optionen ist, dass z die Netzwerkverbindungen aufzeichnet und y die Netzwerkverbindungen ausser acht lässt. Man erkennt den Monitormode an einem 0> Prompt.

Schritt 5: Die einzelnen Konfigurationseinstellungen

0 Server

```
Network mode: 0=Wired Only, 1=Wireless Only, 2=Bridging(One Host) (1) ?
IP Address : (192) .(168) .(000) .(009)
Set Gateway IP Address (N) ?
Netmask: Number of Bits for Host Part (0=default) (0)
Set DNS Server IP addr (N) ?
Change telnet config password (N) ?
```

Die IP Adressen der jeweiligen Bots müssen im gleichen Broadcast Netzbereich (hier 192.168.0.0/24) liegen. Die Bots besitzen alle eine eindeutige Nummer, welche in der Regel mit einem Stift auf das Bodenblech geschrieben wurde. Diese Nummer gibt man am Besten im letzten Ziffernblock der IP Adresse an, um eine Eindeutigkeit der vergebenen IP Adressen zu gewährleisten. In diesem Fall hat der Bot die Nummer 9. Die restlichen Angaben sollten mit den Klammerwerten übereinstimmen.

1 Channel 1

Bei der Konfiguration von Channel 1 ist darauf zu achten, dass die Baudrate dem Standard (9600) entspricht und der Port eine eindeutige Nummer besitzt. Wenn man die in Klammer angegebenen Werte übernimmt befindet man sich auf der sicheren Seite. Geschützte Ports können in der WiPort® Dokumentation[2] von Lantronix nachgelesen werden.

```
Baudrate (9600) ?
I/F Mode (4C) ?
Flow (00) ?
Port No (10001) ?
ConnectMode (CO) ?
Send '+++' in Modem Mode (Y) ?
Show IP addr after 'RING' (Y) ?
Auto increment source port (N) ?
Remote IP Address : (000) .(000) .(000) .(000)
Remote Port (0) ?
DisConnMode (00) ?
FlushMode (00) ?
DisConnTime (00:00) ?:
SendChar 1 (00) ?
SendChar 2 (00) ?
```


2 Channel 2

Die Einstellungen des Channel 2 sind entscheidend für die Kommunikation mittels UDP. Auch hier steht man auf der sicheren Seite, wenn man die angegebenen Werte einfach übernimmt. Der ConnectMode muss auf CC und der Datagram Type auf 01 (Directed oder Broadcast UDP) eingestellt werden. CC bedeutet hierbei, dass alle eingehende Verbindungen angenommen werden, keine Beantwortung stattfindet und Directed UDP als Datagram Type verwendet werden soll.

```
Baudrate (57600) ?  
I/F Mode (4C) ?  
Flow (00) ?  
Port No (10002) ?  
ConnectMode (CC) ?  
Datagram Type (01) ?  
Send as Broadcast (N) ?  
Remote IP Address : (192) . (168) . (000) . (255)  
Remote Port (10002) ?  
Pack Cntrl (00) ?  
SendChar 1 (00) ?  
SendChar 2 (00) ?
```

3 E-mail

Die E-mail Funktionen spielen keine Rolle und werden in dieser Ausarbeitung nicht beachtet.

4 WLAN

```
Topology: 0=Infrastructure, 1=Ad-Hoc (1) ?  
Network name (SSID) (ctbot) ?  
Channel (11) ?  
Security suite: 0=none, 1=WEP (0) ?  
TX Data rate: 0=fixed, 1=auto fallback (1) ?  
TX Data rate: 0=1, 1=2, 2=5.5, 3=11, 4=18, 5=24, 6=36, 7=54 Mbps (3) ?
```

Die WLAN Einstellungen sind weitestgehend selbsterklärend. Die Topologie des Netzwerks ist Ad-Hoc, der Name des Netzwerks muss bei allen Bots ctbot heissen und der Channel auf 11 eingestellt sein. Die Kommunikation der Bots soll unverschlüsselt stattfinden und die Übertragungsrate automatisch zurückfallen, wenn die hier gewählten 11Mbps zu hoch gesetzt wurden.

5 Expert

Die hier gezeigten Einstellungen entsprechen den Standards. Die genauen Auswirkungen der einzelnen Konfigurationsoptionen können in der WiPort® Dokumentation[2] von Lantronix nachgelesen werden.

```
TCP Keepalive time in s (1s - 65s; 0s=disable): (45) ?  
ARP Cache timeout in s (1s - 600s) : (600) ?  
CPU performance (0=Regular, 1=Low, 2=High): (0) ?  
Disable Monitor Mode @ bootup (N) ?  
HTTP Port Number : (80) ?  
SMTP Port Number : (25) ?  
MTU Size (512 - 1400): (1400) ?  
Enable alternate MAC (N) ?  
Ethernet connection type: (0) ?
```

6 Security

Die Sicherheitseinstellungen werden hier nur der Vollständigkeit Halber gelistet. Die Änderung dieser Konfigurationsoptionen sollten gut überlegt werden. Es ist generell verboten Passwörter zu setzen.

```
Disable SNMP (N) ?
SNMP Community Name (public):
Disable Telnet Setup (N) ?
Disable TFTP Firmware Update (N) ?
Disable Port 77FEh (N) ?
Disable Web Server (N) ?
Disable Web Setup (N) ?
Disable ECHO ports (Y) ?
Enable Enhanced Password (N) ?
Disable Port 77F0h (N) ?
```

7 Defaults

Das Herstellen der Werkseinstellungen bietet sich in der Regel vor dem Beginn der Konfiguration des Bots an.

8 Exit without save

Selbsterklärend

9 Save and exit

Selbsterklärend

Schritt 6: Das Webinterface testen

Es gibt verschiedene Wege den Roboter zu konfigurieren. WiPort® bietet eigens hierfür ein Webinterface. Wenn man Schritt 1 bis Schritt 5 wie zuvor beschrieben durchgeführt hat, kann man sich nach einem Neustart des CTBot von seinem Rechner aus dem Netzwerk ctbot anschliessen. Bis der eigene Rechner das WLAN Netzwerk findet, können bis zu 60 Sekunden vergehen. Hat man sich dem Netzwerk angeschlossen, so erreicht man die Konfigurationsoberfläche unter der IP, welche man bei der Konfiguration (0 Server) vergeben hat (z.B. <http://192.168.0.9>). Der Benutzername lautet Admin. Passwort wurde ja keines gesetzt. Sollte man sich nicht mit diesen Einstellungen anmelden können sollte man zu aller erst die Ausgangskonfiguration (7 Defaults) wieder herstellen und danach Schritt 1 bis Schritt 5 erneut durchgehen. An dieser Stelle sei noch einmal auf die exzellente Dokumentation[2] des WiPort® Moduls von Lantronix hingewiesen.

Das Master/Slave Modul

Was bedeutet Master/Slave?

Wikipedia[4] liefert dazu folgende Definition: „Der Begriff Master/Slave (dt. Herr/Sklave) bezeichnet eine Form der hierarchischen Verwaltung des Zugriffs auf eine gemeinsame Ressource in zahlreichen Problemstellungen der Regelung und Steuerung.“

Warum heisst es Master/Slave Modul?

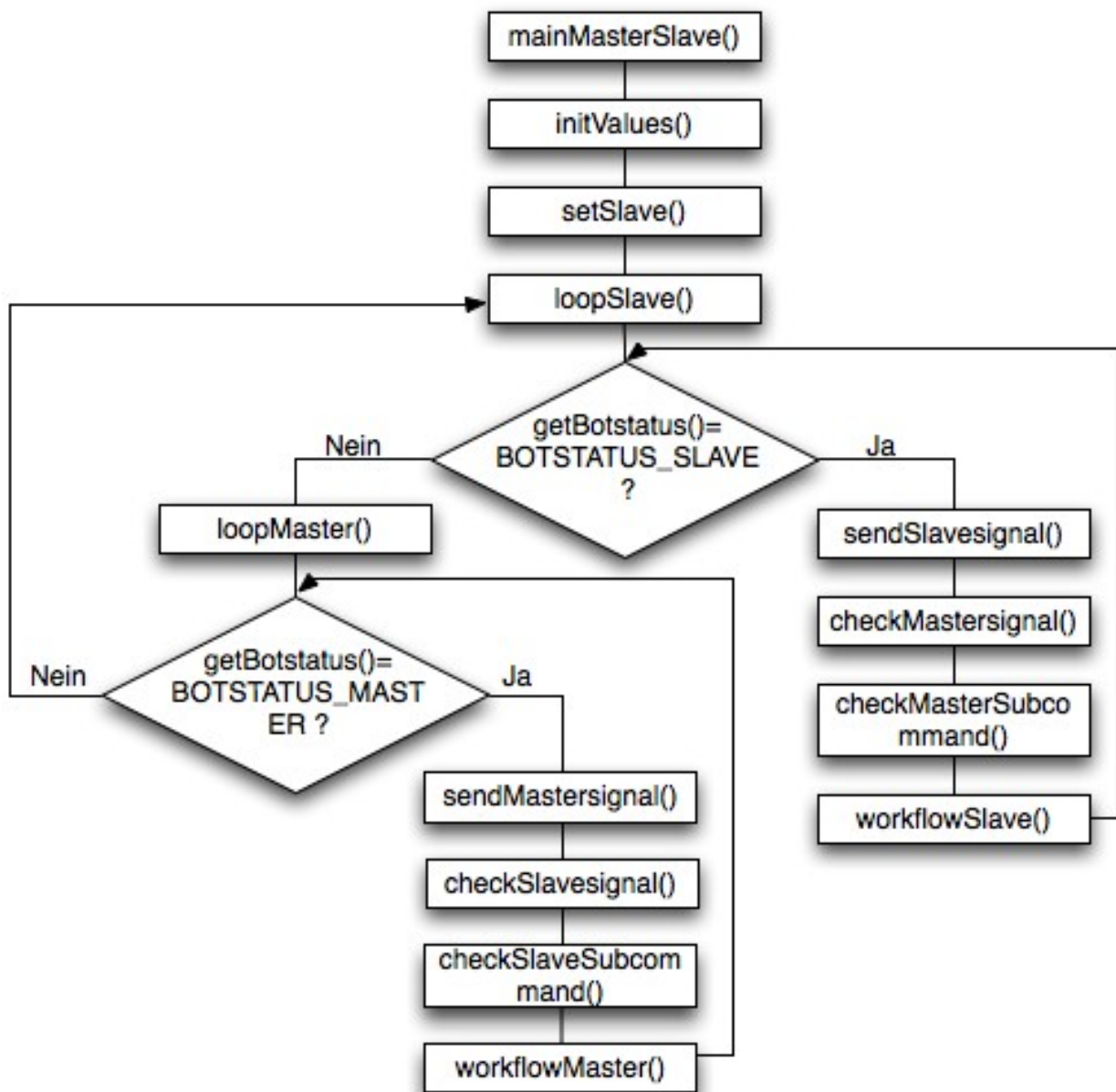
Von einem Modul spricht man in der Softwaretechnik von einer in sich abgeschlossenen Komponente für eine definierte Aufgabe. Bei Modulen sollte eine Trennung von Schnittstelle und Implementierung stattfinden und genau dies geschieht hier in abstrahierter Form. Man hat auf der einen Seite eine ganze Reihe von Funktionen, welche das gesamte Master/Slave Verhalten steuern und somit eine separate Programmlogik bilden, welche nicht mehr modifiziert werden muss. Auf der anderen Seite gibt es nun lediglich 4 Funktionen, welche der Programmierer bei der Implementierung neuer Aufgaben anpassen muss, was eine Einbindung weiterer Module vereinfacht. Die 4 Funktionen, um welche sich man noch kümmern muss heissen im Einzelnen workflowMaster, workflowSlave, checkMasterSubcommand und checkSlaveSubcommand. Die Verarbeitung von Aufgaben erfolgt dann gänzlich über Kommandos, welche im folgenden Abschnitt ausführlich beschrieben werden.

Kommandos und Subkommandos – SIGNALS und SUBSIGNALS

Primärkommandos bzw. Subkommandos werden im Master/Slave Modul SIGNALS bzw. SUBSIGNALS genannt. Es bleibt nicht aus, geeignete und eventuell eindeutige Werte für Subkommandos festzulegen. Subkommandos sind ebenso wie die Primärkommandos auf uint8 festgesetzt, also können maximal 256 (0-255) unterschiedliche Werte für Subkommandos definiert werden. Die Headerdatei master_slave.h beinhaltet bereits eine Reihe von definierten Subkommandos (z.B. #define SUBSIGNAL_STARTSEARCH_LIGHT) und sollte auch ausschliesslich in dieser Datei erweitert werden. Im Gegensatz zu den Subkommandos wurden die Primärkommandos als Shift Operationen definiert. So ist es zum Beispiel theoretisch möglich, einem Bot den Status Master und Slave gleichzeitig zu geben und diesen auch explizit wieder abzurufen, was in der Praxis natürlich wenig Sinn macht. Es ergeben sich aber durchaus sinnvolle Anwendungsmöglichkeiten, wie das folgende Beispiel verdeutlichen soll. Falls 256 Subkommandos für die Erteilung von Befehlen nicht ausreichen kann ein neues Primärkommando (z.B. #define SIGNAL_MASTER_2 1<<3) implementiert werden, um im Anschluss weitere 256 Subkommandos verwenden zu können. Die Funktion checkMastersignal() muss dann natürlich dementsprechend angepasst werden. Die Stelle an der dies geschehen kann wurde im Quellcode gekennzeichnet. Verwaltet werden die Signale bzw. Kommandos innerhalb der Funktionen checkMasterSignal(), checkSlaveSignal(), checkMasterSubcommand() sowie checkSlaveSubcommand(). Die beiden Funktionen workflowMaster() und workflowSlave() dienen dann zur Abarbeitung der gegebenen Befehle. Eine genaue Beschreibung der Funktionen können in der jeweilige Funktionsbeschreibung nachgelesen werden.

Die Funktionsweise des Master/Slave Moduls

Folgendes Schaubild hat keinen Anspruch auf Vollständigkeit und soll ausschliesslich die Funktionsweise des Master/Slave Moduls verdeutlichen.



Wie im Kapitel Kommandos und Subkommandos – SIGNALS und SUBSIGNALS bereits beschrieben wurde, werden SIGNALS bzw. SUBSIGNALS für das Setzen eines bestimmten Status verwendet. Ein Status kann hierbei z.B. für eine Aufgabe stehen, welche ein Slave ausführen muss, oder der Zustand, in welchem sich der Bot befindet. Im hier verwendeten Modul sind genau zwei Zustände definiert, welche der Roboter einnehmen kann, um seinen eigenen Status zu markieren. Diese beiden Zustände `BOTSTATUS_SLAVE` bzw. `BOTSTATUS_MASTER` werden der statischen Variable `botstatus` zugewiesen und bei Bedarf verglichen. Ein dritter, undefinierter Zustand wurde durch die Tatsache unnötig, dass jeder Bot mit hoher Wahrscheinlichkeit Slave ist und diesen Status mit der Funktion `setSlave()` in Anspruch nimmt. Sollte sich diese Annahme als falsch herausstellen, so reisst sich der vermeintliche Master nach einer zufälligen Zeitspanne in der Funktion `checkMastersignal()` die Macht an sich und wechselt

beim nächsten Vergleich von `getBotstatus() = BOTSTATUS_SLAVE` in die Funktion `loopMaster()`. Sollte der ungünstige Fall auftreten, dass sich ≥ 2 Bots gleichzeitig zum Master hochstufen, wird um den Mastertitel gestritten. Natürlich wird nicht wirklich um den Mastertitel gestritten, sondern getreu dem Motto „der klügere gibt nach“ setzt sich der Bot zurück in den Slavemodus, welcher ein Mastersignal empfängt. Sollte daraufhin nur noch ein Bot übrig sein, welcher den Masterstatus innehält, läuft alles weiter wie gewohnt. Falls sich dahingegen kein Bot mehr als Master berufen fühlt greift sich der nächste Slave, wessen Timeout heruntergelaufen ist den Mastertitel. Um Verklemmungen zu vermeiden werden nach jedem Durchlauf eines solchen Szenarios zeitverzögert, neue Zufallswerte für das Timeout gesetzt. Das Timeout von welchem hier gesprochen wird steht für das Zeitintervall, in welchem sich ein Master bei den Slaves gemeldet haben muss. Dieser Wert muss ausreichend gross gesetzt werden, um Problemen wie Funklöchern etc. entgegenzuwirken.

Funktionsbeschreibung

Die Beschreibung der einzelnen Funktionen können direkt im Quellcode nachgelesen werden. Im Folgenden werden alle bisher implementierten Funktionen der Quellcodedatei **`master_slave.c`** gelistet.

`void workflowSlave(void);`

Arbeitsanweisungen für den Slave Bot.

`void workflowMaster(void);`

Arbeitsanweisungen für den Master Bot

`void addBotId(int16, int16);`

Fügt einen Bot zur Liste hinzu. Die Liste wird ausschliesslich vom Master Bot verwaltet.

`void deleteBotId(int16);`

Löscht einen Bot aus der Liste. Die Liste wird ausschliesslich vom Master Bot verwaltet.

`int16 searchBotId(int16, int16);`

Suche nach einem Bot. Wird vom Master Bot aufgerufen.

`int16 countBots(void);`

Anzahl der Slaves ermitteln. Wird vom Master Bot aufgerufen.

`uint8 getBotstatus(void);`

Funktion liefert den momentan gesetzten Status Master oder Slave.

`int16 createRandomID(void);`

Funktion erzeugt eine zufällige ID, welche einen Bot identifiziert.

`void setRandomValue(void);`

Funktion setzt einen Zufallswert.

`uint16 getRandomValue(void);`

Funktion holt sich den gesetzten Zufallswert.

`void setSlave(void);`

Bot setzt seinen Status auf Slave.

`void setMaster(void);`

Bot setzt seinen Status auf Master.

`void setMastersignalCountdown(uint16);`

Funktion setzt den Countdown zum Senden des Mastersignals.

`uint16 getMastersignalCountdown(void);`

Funktion liefert den Wert des Mastersignals.

`void decMastersignalCountdown(void);`

Funktion zählt den Countdown des Mastersignals um eins herunter.

`void setSlavesignalCountdown(uint16);`

Funktion setzt den Countdown zum Senden des Slavesignals.

`uint16 getSlavesignalCountdown(void);`

Funktion liefert den Wert des Slavesignals.

void decSlavesignalCountdown(void);

Funktion zählt den Countdown des Slavesignals um eins herunter.

void sendMastersignal(void);

Funktion sendet via UDP ein Mastersignal.

void sendSlavesignal(uint16);

Funktion sendet via UDP ein Slavesignal.

void loopMaster(void);

Die Hauptschleife des Master Bots.

void loopSlave(void);

Die Hauptschleife des Slave Bots.

void checkMasterSubcommand(command_t);

Funktion wertet eventuelle Subkommandos aus, welche vom Master Bot gesendet wurden.

void checkMastersignal(void);

Funktion überprüft das Mastersignal.

void checkSlaveSubcommand(command_t);

Funktion wertet eventuelle Subkommandos aus, welche vom Slave Bot gesendet wurden.

void checkSlavesignal(void);

Funktion überprüft das Slavesignal.

void initValues(void);

Funktion initialisiert wichtige Variablen des Bots.

void mainMasterSlave(void);

Funktion bildet den Einstiegspunkt des Master/Slave Moduls.

Abschluss

Zum Abschluss will ich hier noch ein paar Worte loswerden im Bezug auf den heutigen Stand (also zwei Tage vor Abgabefrist). Ich habe in der letzten Woche noch einige Änderungen am Quellcode vorgenommen, welchen ich leider nicht mehr so ausgiebig testen konnte, wie ich das wollte. Die letztendliche Version, welche ich abgebe ist Version 8 und kann neben Version 7 im Ordner Sicherungen gefunden werden. Die Roboter 1, 8 und 9 wurden von mir vor wenigen Augenblicken nochmals geflasht und für gut (funktionierend) empfunden. Leider ist der Lichtsuche etwas arg spärlich von mir implementiert, da ich mein Hauptaugenmerk auf die Kommunikation und der Aushandlung des eigentlichen Master/Slave-Modus der Bots legte. Als wenig hilfreich habe ich die doch beachtliche Sendeleistung der Wlan Module empfunden. Bei meinen letzten Tests wollte ich unbedingt erzwingen, dass es zwei Master Bots gleichzeitig gibt und wie das genaue Verhalten ist, wenn sie aufeinander treffen. Wenn man gerne wandert, wird man diese Tests sicherlich gerne machen. Bugs in diesem Punkt kann ich deshalb nicht ausschliessen. Da ich mir aber fest vorgenommen habe am Thema zu bleiben, wird sich dafür bestimmt noch eine Möglichkeit finden, wie man dieses „Wanderproblem“ umgehen kann. Sämtliche Intervalle sind deutlich zu hoch gefasst worden von mir und können natürlich Schrittweise nach unten angepasst werden. Im Vergleich zu der ersten Erfahrung im Grundstudium, als mein Team unglücklicherweise einen dermassen zerstörten Bot bekommen hat, wo man dann wirklich bis auf Hardwareprobleme lösen bzw. lösen lassen zu nichts weiter kam, machte es dieses mal richtig Lust auf mehr. Die billigen Lichtsensoren machen nach wie vor Ärger. Ich habe an 2 Bots 2 Lichtsensoren mit gebrauchten Sensoren getauscht, wobei einer immer noch Faxen macht. Welcher Sensor was taugt und welchen man besser mal bei Gelegenheit tauschen sollte kann man während der Initialisierung sehen. Was ist noch zu sagen? Falls ihr noch auf der Suche nach einem Wahlpflichtfach seit, kann ich euch die Weiterentwicklung nur empfehlen. Top Arbeitsbedingungen und schnelles Feedback bei Problemen vom Professor bzw. anderer Studenten brachten mir zumindest ein gutes Gefühl bei der Arbeit. Weniger positiv war der Totalausfall des zweiten Studenten, welcher eigentlich auch an diesem Projekt teilnehmen wollte. Wenn jemand noch Fragen oder Anregungen hat, kann er oder sie sich gerne mit mir in Verbindung setzen. In diesem Sinn: Danke fürs Lesen.

Literatur

Hinweis

Die Quellen lassen sich auch im Ordner C:\CTBot\documents der VMware finden.

Quellen

- [1] <http://www.heise.de/trac/ctbot/wiki/SVNUndEclipse>
- [2] http://www.lantronix.com/pdf/WiPort_UG.pdf
- [3] <http://www.vmware.com/de/>
- [4] <http://de.wikipedia.org/>
- [5] Mobile Roboter – Dokumentation für Mobile Roboter im Hauptstudium – 30.07.2008