

Besondere Lernleistung

Computergestützte
Fahrzeugsteuerung

Solarcup
2012/2013

Gliederung

1. Warum dieses Thema?
2. Funktionsprinzip
3. Hardware
 - 3.1 Controller
 - 3.2 CarPC
 - 3.3 Sensoren
 - 3.4 Fahrregler
 - 3.4.1 PWM
4. Software
 - 4.3 Klasse CError (Fehlerdokumentierung)
 - 4.2 Server(Fahrzeug)
 - 4.2.1 Klasse CMotor
 - 4.3 Client(Steuerungssoftware/Controller)
5. Übertragung
 - 5.1 IEE802.11G
 - 5.2 Warum getrennte Sockets für jeden Motor?
6. Ausfallsicherheit/Sicherheitsrisiken
7. Bau und Realisierung

1. Warum dieses Thema?

Für mich war dieses Thema ideal, da ich hier meine Kenntnisse praktisch anwenden und hervorragend erweitern konnte. Dieses Thema verbindet weiterhin Elektronik und Informatik. Das sind Schwerpunkte die ich auch im späteren Berufsleben weiterverfolgen möchte. Ich denke, dass man durch den strategisch sinnvollen Einsatz von Computertechnik im Solarauto eine gute Steigerung von sowohl den Fahreigenschaften (Fahrstabilität, Geschwindigkeit, Wendigkeit) als auch der Ökonomieerzielen kann. Dies ist auch gleichzeitig eine gute Gelegenheit meine Gedanken und Ideen in der Praxis zu testen.

Vielen Dank nochmals an:

Die Max-Eyth-Schule Kassel (weil sie mir diese Möglichkeit sowie technisches Material und Knowhow gab)

Frau Müller (als Betreuungs-Lehrerin und als Ansprechpartner)

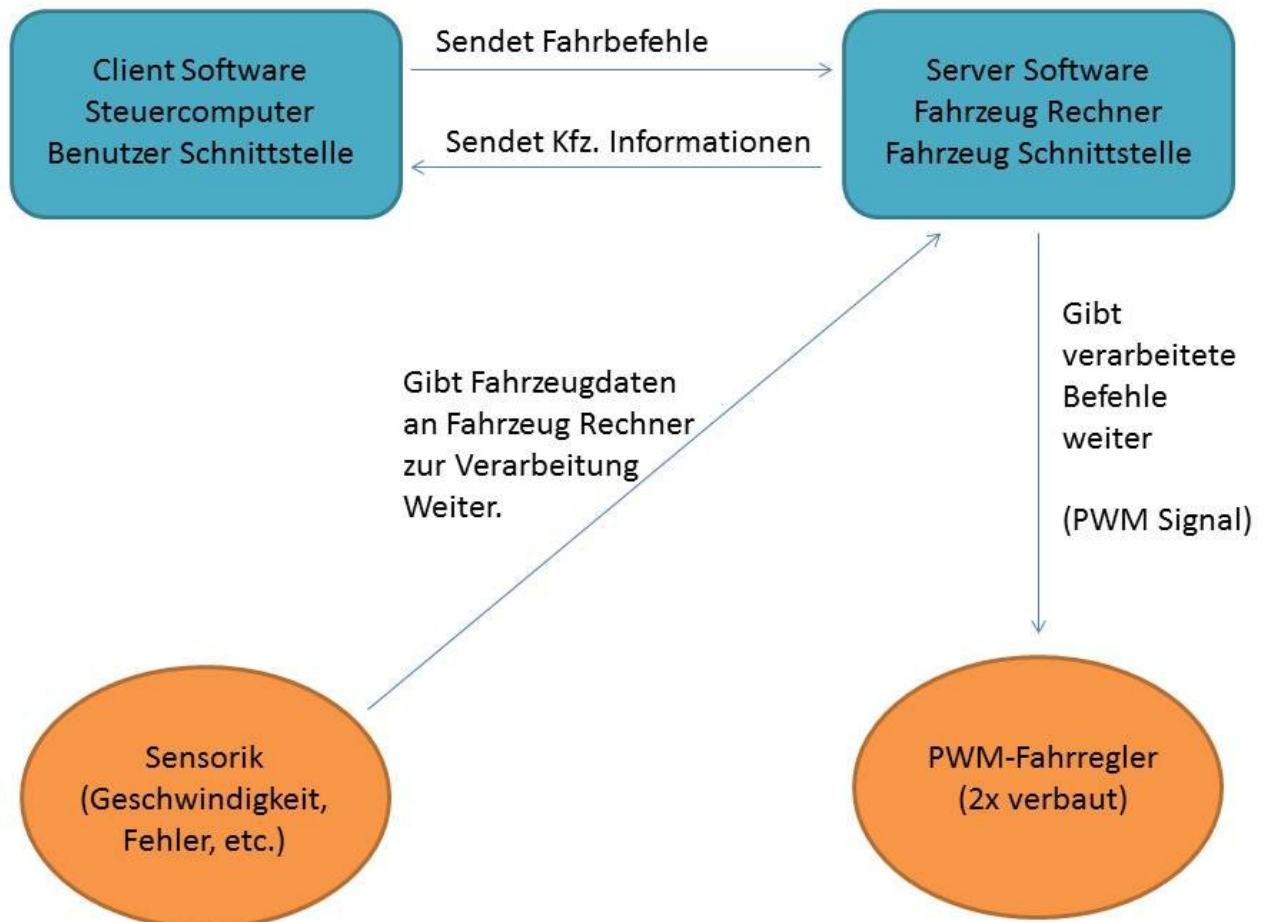
Herr Behrend (ebenfalls als Beratungslehrer und wegen technischer Unterstützung)

Unsere beiden Teams (wir schaffen das dieses Jahr ;))

Uni Kassel/Solarcup (wegen dem jährlich stattfindenden Solarcup und Hilfe bei technischen Fragen)

2. Funktionsprinzip

Im Solarauto ist ein kleiner Bordcomputer verbaut, die darauf installierte Serversoftware empfängt Daten von der Fernsteuerung (Clientsoftware). Diese Daten beinhalten, für jeden Motor einzeln, Geschwindigkeitswerte oder Stoppbefehle. Der Bordrechner steuert dann den jeweiligen Fahrregler mit der entsprechenden Geschwindigkeit an (PWM-Signal). Der Bordcomputer könnte mit Hilfe der an ihn angeschlossenen Sensorik auch autark fahren, dies ist aber noch nicht vorgesehen. Der Bordrechner sendet stattdessen die Daten an den Controller der diese dann auswertet.



3. Hardware

3.1 Controller

Als Controller dient entweder ein Laptop mit oder ohne Joystick oder ein Android Tablet (App noch in Entwicklung).

Der Controller hat eine Steuersoftware, die je nach Neigungswinkel des Tablet/Joysticks einen prozentualen Wert für die Motordrehzahl jedes der beiden Motoren errechnet.

Beispiel: Voller Rechtseinschlag Motor 1:25% Motor 2:80%. Volle Fahrt geradeaus Motor 1:100% Motor 2:100%.

Diese Werte werden alle 10 Millisekunden auf 2 getrennten Sockets an das Auto gesendet. Des Weiteren werden alle 5 Sekunden vom Auto die Sensordaten (Geschwindigkeit, Fahrwerksfehler etc.) empfangen.

3.2 CarPC

Im Solarauto ist ein Mikrocomputer eingebaut.

Verwendet wird ein Via Mainboard mit 1GHz VIA CPU.

Da die CPU eine x86 Architektur verwendet, wird als System Microsoft Windows XP Embedded verwendet. Dieses ist Ressourcen schonend und bringt alle Voraussetzungen mit, um mit der Software sowie den über die serielle Schnittstelle angesteuerten Fahrreglern zu interagieren.

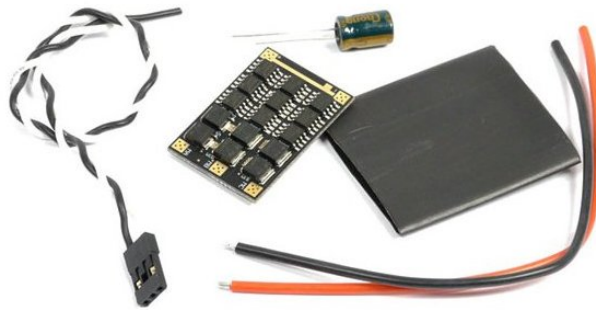
Das System besitzt 2 physikalische serielle Anschlüsse (RS232) sowie WLAN IEEE 11E.BGN und Gigabit LAN.

Der Rechner sollte ursprünglich ein Adhoc WLAN Netz hosten, dies wurde jedoch im Hinblick auf die schlechte Konfigurierbarkeit und die verhältnismäßig schlechte Verbindungsqualität verworfen. Sowohl das Fahrzeug als auch der Steuercomputer verbinden sich jetzt auf einem Accespoint, der im Mannschaftszelt aufgestellt wird.



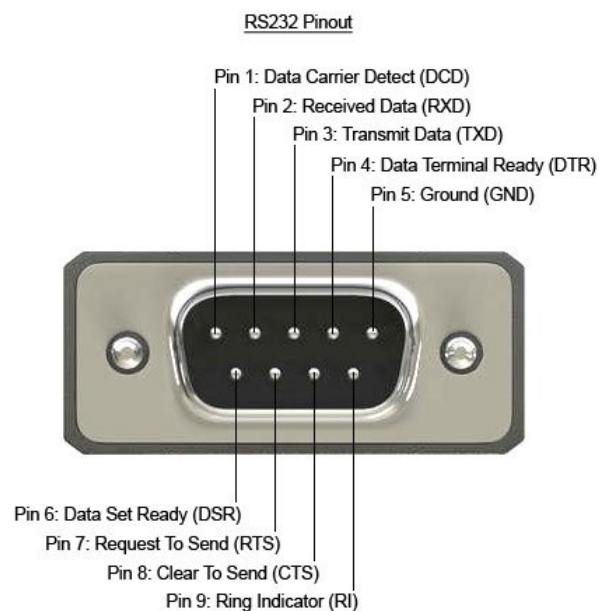
3.4 Fahrregler

Da wir ein PWM-Signal zur Verfügung haben war es uns nicht möglich einen Standard Modellbau Fahrregler zu verwenden. Es war geplant selber einen Regler mit einer Transistorschaltung zu konstruieren. Nach einigen fehlgeschlagenen Versuchen wurde dieses Vorhaben jedoch aufgrund der nicht allzu großen Zeitspanne bis zur Fertigstellung des Projektes verworfen. Als neue Fahrregler fungieren nun zwei PWM gesteuerten Modellflug Fahrreglern von Flyduino. (<http://flyduino.net/Flyduino-30A-ESC>)



Diese Fahrregler werden mit einem 1KHz PWM-Signal angesteuert, sie haben eine hohe Belastbarkeit, einen hohen Wirkungsgrad und sind kostengünstig (20,90€).

Die Ansteuerung erfolgt über den RTS Pin am seriellen Anschluss am Fahrzeugrechner.



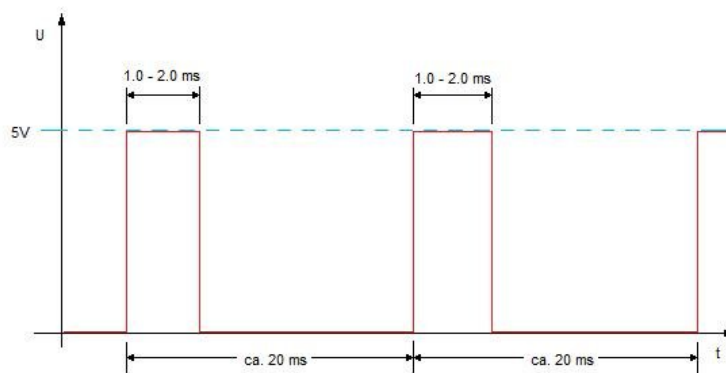
Als Beispiel zur Ansteuerung dient diese Funktion.

(MySerial ist der serielle Anschluss, m_iGeschwindigkeit ist die eingegebene, prozentuale Motorgeschwindigkeit)

```
Void setSpeed(int m_iGeschwindigkeit)
{
    Int pause=(1000-m_iGeschwindigkeit*10)\2;
    // Hier wird der Pausenintervall ausgerechnet
    MySerial->setRTS(1);
    Sleep(pause);
    MySerial->setRTS(0);
    Sleep(pause);
}
```

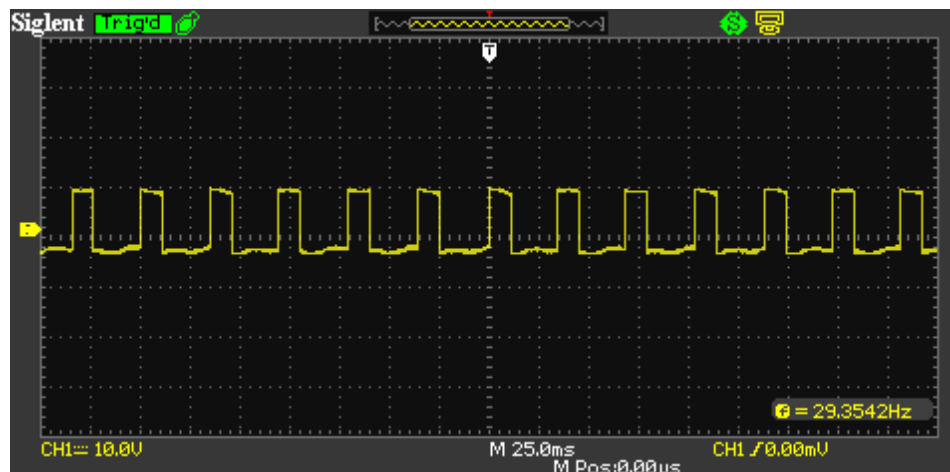
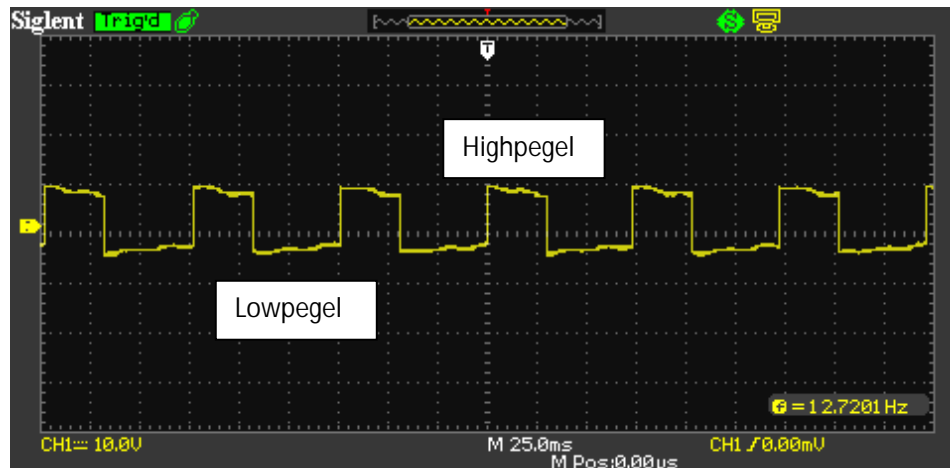
Diese Funktion wird als Schleife in einem eigenen Thread ausgeführt. Die Berechnung der Pause erfolgt auf der logischen Grundlage, dass wir für 100% Geschwindigkeit eine Pause von 1/1000 Sekunden benötigen und für 0%Geschwindigkeit eine Pause von 1 Sekunde. Da wir 2 Sleep Befehle verwenden, müssen wir das Ergebnis noch durch zwei teilen.

Ein Standard Modellbau Fahrregler konnte auf Grund seines Signalaufbaus nicht verwendet werden. Hier wird ein 20 Millisekunden Zeitfenster benutzt. Je nachdem wie lange der Highpegel am Anfang dieses Zeitfensters gesetzt wird (0-2 Millisekunden) wird die Geschwindigkeit erzeugt.



3.4.1 PWM

Pulsweitenmodulation (Englisch pulsewidth modulation=PWM) ist eine Modulationsart bei der die Spannung einen wechselnden Wert einnimmt. In unserem Falle wird PWM zur Ansteuerung des Fahrreglers verwendet. Das Signal was wir erhalten besteht aus jeweils einem Highpegel (9V DC) und einem Lowpegel (0V DC). Beide Phasen haben (annähernd) dieselbe Zeitdauer. Der Fahrregler errechnet die an den Motor abgegebene Leistung anhand der Wechselfrequenz zwischen High- und Lowpegel. Hierbei ist eine Wechselfrequenz von 1 KHz (1000 Änderungen pro Sekunde) 100% Motorleistung. Das obere Bild stellt ein PWM-Signal zu einer langsamen Geschwindigkeit (12Hz) dar. Das untere ein Signal zu einer hohen Geschwindigkeit (29Hz).



4 Software

Die Software im Auto (Server) sowie die Software auf der Fernsteuerung (Client) sind komplett von mir entworfen und implementiert. Sie ist vollständig in C/C++ geschrieben. Im Client wird in späteren Versionen die Microsoft Foundation Class (MFC) für eine GUI verwendet werden. Weiterhin verwende ich die angepassten Zentralabitur Klassen CServerSocket, CSocket und CSerial. Diese Klassen dienen zur einfachen Verwendung von Sockets und der seriellen Schnittstelle. Als Programmierumgebung wurde Microsoft Visual Studio 2010 verwendet.

4.1 CError

Die Klasse CError dient der Fehlerausgabe. Sie ist für die Funktionalität nicht essentiell, vereinfacht jedoch den Programm Code da bestimmte Fehler an mehreren Stellen im Programm auftauchen können. So muss nur einmal jede Fehlerausgabe programmiert werden. In späteren Versionen soll die Fehlerausgabe zur besseren Debugmöglichkeit auch über die Datenleitung eines der RS232 Ports ausgegeben werden könne.

Eine Beispiel Funktion bei einem Fehler im Zusammenhang zum seriellen Port:

```
void CError::ComErr()
{
    cout<<endl;
    cout<<"#####" <<endl;
    cout<<"#                               #" <<endl;
    cout<<"#////////Kritischer FEHLER beim oeffnen des Com Portes////////#" <<endl;
    cout<<"#                               #" <<endl;
    cout<<"#####" <<endl;
    system("pause");
    exit(1); //An dieser Stelle wird das komplette Server Programm beendet
}
```


4.2.1 Fahrzeug (Server)


Auf dem Bordrechner des Fahrzeugs läuft die Server Software. Sie stellt den ServerSocket und die Verbindung zum seriellen Anschluss zur Verfügung.








4.2.2 Die Klasse CMotor (Version 0.3.1)


CMotor







Class



 **Felder**

-  ComPort
-  MotorID
-  MyCom
-  MyServer
-  MySock
-  NwPort
-  Speed

 **Methoden**

-  ~CMotor
-  CMotor
-  init
-  RecvSpeed
-  SendState
-  SetSpeed

Die Klasse CMotor enthält als Attribute:

- ComPort: String //Name des ComPortes
- MotorID: Int //ID des Motors zur Identifizierung
- MyCom: CSerial //Objekt der Klasse CSerial zur Ansteuerung des RS232 Portes
- MyServer: CserverSocket //Objekt für den ServerSocket
- MySock: CSocket //Arbeitssocket
- NwPort: Int //Port über den der Motor im Netzwerk erreichbar ist
- Speed: int //Aktuelle Geschwindigkeit als prozentualer Wert

Zusätzlich werden drei Threads benötigt und ausgeführt um das simultane Empfangen, senden von Daten sowie das Senden der Befehle an den Fahrregler ermöglichen.

Die Klasse enthält folgende Funktionen:

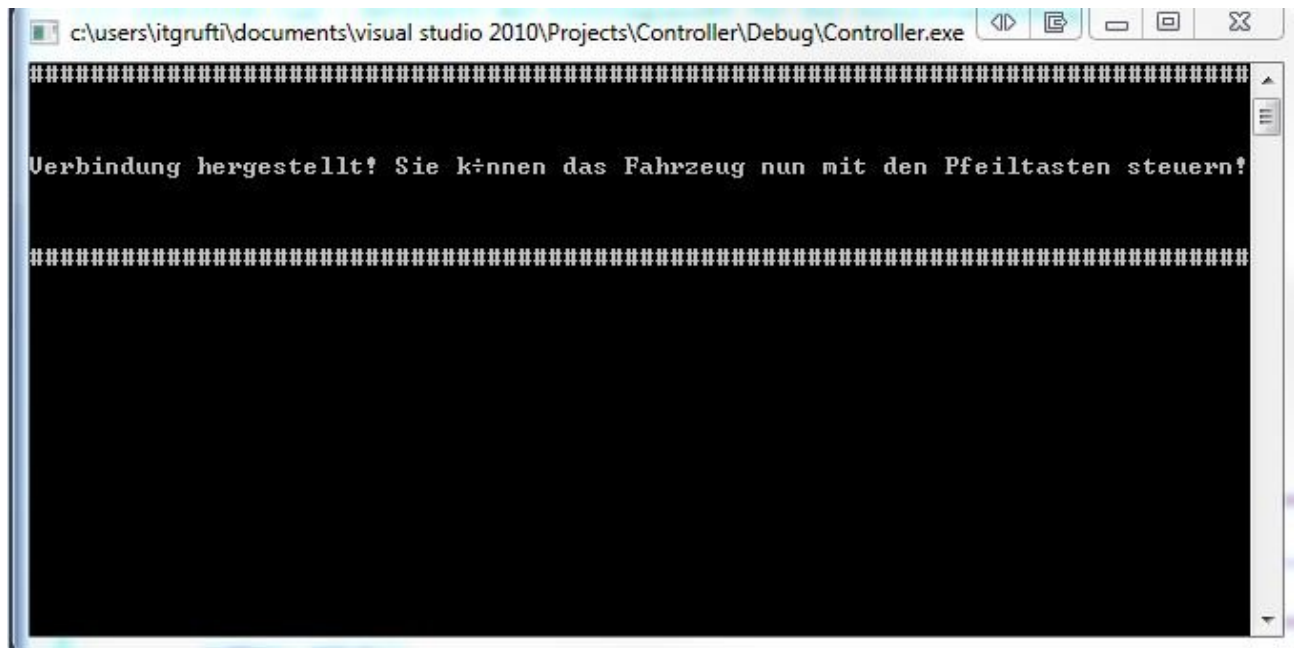
- CMotor(int ID, int NwPort, string ComPort) //Konstruktor, Belegung der Variablen
- Init() //Öffnen des ComPorts, Warten auf Netzwerkverbindung (Initialisierung)
- RecvSpeed() //Lesen der neuen Geschwindigkeit vom Socket
- SendState() //Senden der Sensordaten an den Client
- SetSpeed() //Senden der Daten an den Fahrregler (RS232)

4.3 Client (Steuerungssoftware/Controller)

Auf dem Steuerungslaptop läuft die Clientsoftware. Diese verbindet sich mit dem ServerSocket des Autos. Sie empfängt die Sensordaten des Autos und wertet diese aus (siehe Ausarbeitung von Florian Gaiser).

Die gewünschten Geschwindigkeiten werden vom Client durch den Joystick ausgelesen und in zwei prozentuale Werte umgerechnet die dem Auto per TCP/IP Verbindung gesendet werden.

In der Steuerungssoftware wird die Klasse CError zu Fehlerausgabe sowie die Zentralabitur Klasse CSocket verwendet. In der momentanen Version werden noch die Pfeiltasten zu Steuerung verwendet. Die vollständige Software befindet sich im Anhang.



```
c:\users\jgrufti\documents\visual studio 2010\Projects\Controller\Debug\Controller.exe
#####
Verbindung hergestellt! Sie können das Fahrzeug nun mit den Pfeiltasten steuern!
#####
```

5. Übertragung

Das Fahrzeug ist über WLAN mit dem Steuercomputer verbunden. Die Verbindung ist durch TCP/IP Sockets realisiert. Da jeder Motor einen eigenen Socket hat ist die Übertragung verhältnismäßig simpel. Der Client überträgt eine Ganzzahl zwischen 0 und 100. Dies stellt die Geschwindigkeit des Motors als Prozentwert dar. Der Server überträgt eine Ganzzahl an den Client. Diese Ganzzahl ist die Anzahl der Raddrehungen Pro 4 Sekunden.

5.1 IEEE802.11G

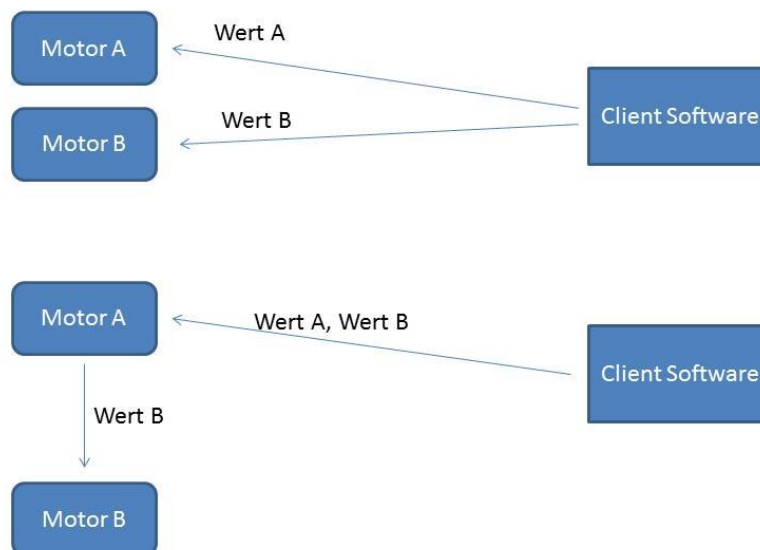
Die Übertragung findet auf dem 2,4 GHz Band statt. Es werden Kanäle in 20 MHz Abstand verwendet. In Europa sollten nur noch die Kanäle 1, 5, 9 und 13 verwendet werden. Die maximale Sendeleistung ist hier auf 125 mW beschränkt. In unserem Fall ist die Verbindung über WPA2 verschlüsselt. Für das Projekt eignet sich die Übertragung via WLAN am besten, da dieser Standard unter einer Windows Umgebung einfach anzuwenden ist. Es entfällt eine Implementierung in den Programmcode.



Wir verwenden eine TP-Link PCI WLAN-Karte im Auto. Diese Karte hat ein gutes Preis/Leistungs-Verhältnis und eine gute Reichweite. Dies ist gerade für eine Fernsteuerung wichtig.

5.2 Warum getrennte Sockets für jeden Motor?

Die Entscheidung jedem Motor einen eigenen Socket zuzuordnen fiel aus zwei Gründen. Zum einen ist so die Skalierbarkeit (mehr als zwei Motoren) gegeben, zum anderen wird so die Steuerverzögerung möglichst gering gehalten. Die Übertragung von zwei Geschwindigkeitswerten parallel ist doppelt so schnell wie das Übertragen von zwei Werten hintereinander. Des Weiteren wird dadurch das Protokoll übersichtlicher da man besser unterscheiden kann welcher übertragene Wert zu welchem Motor gehört.



6. Ausfallsicherheit/Sicherheitsrisiken

Sicherheitsrisiken wenn die Steuerung ausfällt oder der Funkkontakt zum Auto abbricht:

- Das Auto fährt unkontrolliert weiter
- Es könnten Menschen dadurch verletzt werden
- Das Auto nimmt Schaden
- Eventuelle Überlastung der Motoren und Fahrregler
- Schaden an anderen Autos
- Schlechte Chancen im Rennen
- Disqualifikation

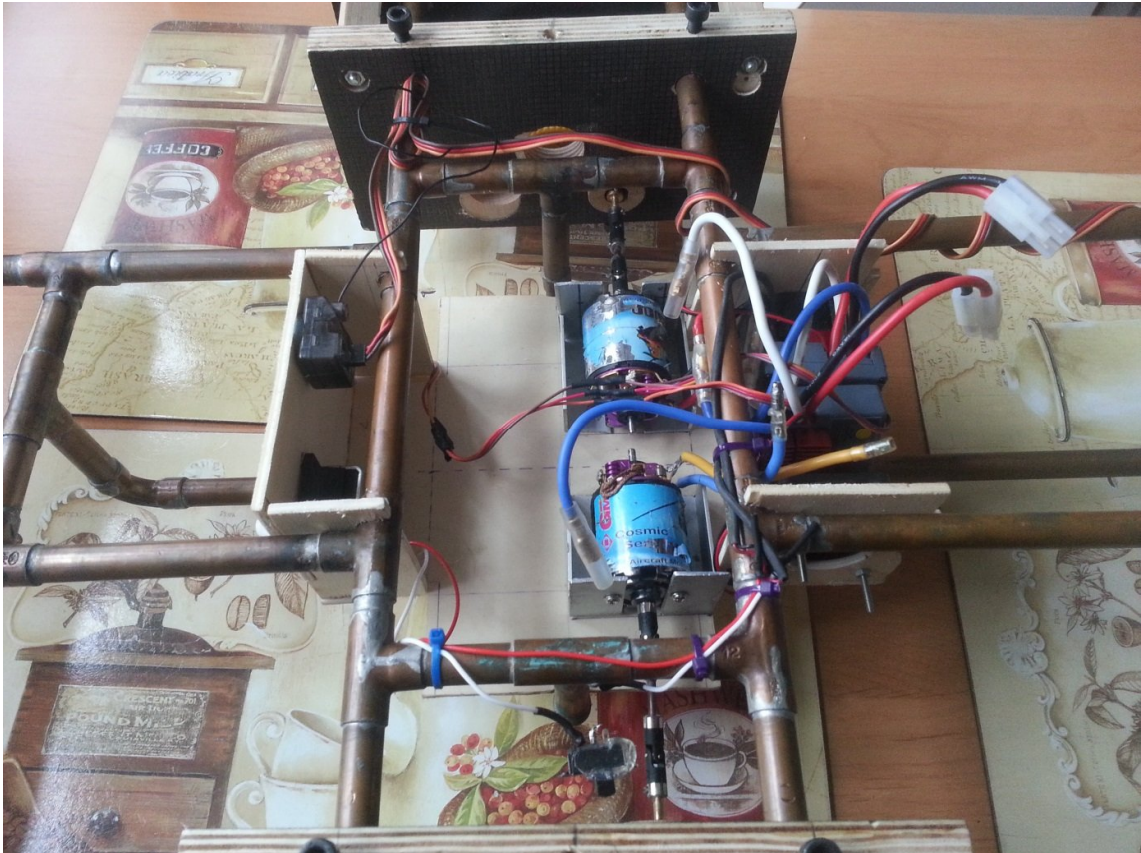
Zur Verbesserung der Ausfallsicherheit wurden folgende Maßnahmen ergriffen:

- Es wird eine WPA2 Verschlüsselung bei der Verbindung verwendet
→Kein Hacken der Verbindung zwischen Steuerung und Auto möglich
- Wenn die Verbindung abbricht beendet sich das Server Programm
→Der Fahrregler kriegt keine Geschwindigkeitsimpulse mehr
- Stürzt die Software ab, liegt kein Signal mehr am Fahrregler an
→Sofortiger Stopp der Motoren
- Es wird ein Accespoint anstatt einer Adhoc Verbindung zu Auto verwendet
→Schnellerer und automatischer Verbindungsaufbau, höhere Leistung
- Bei einem Fehler in der Software sind die Fahrregler sofort abgeschaltet
- Überprüfung der Fahrzeug Technik durch Sensoren

Darüber hinaus entsteht durch die hohe Wendigkeit des Fahrzeugs ein gewaltiger Sicherheitsvorteil, da schnell reagiert werden kann z.B. bei Hindernissen oder anderen Fahrzeugen.

8. Bau und Realisierung

Als Grundgerüst für unser Solarauto verwenden wir den Kupferrahmen des letzten Jahres. Dieser ist ideal für die teure Elektronik da er viel Sicherheit bietet. Es werden neue Motoren, Fahrregler und die neue Steuerungselektronik verwendet.



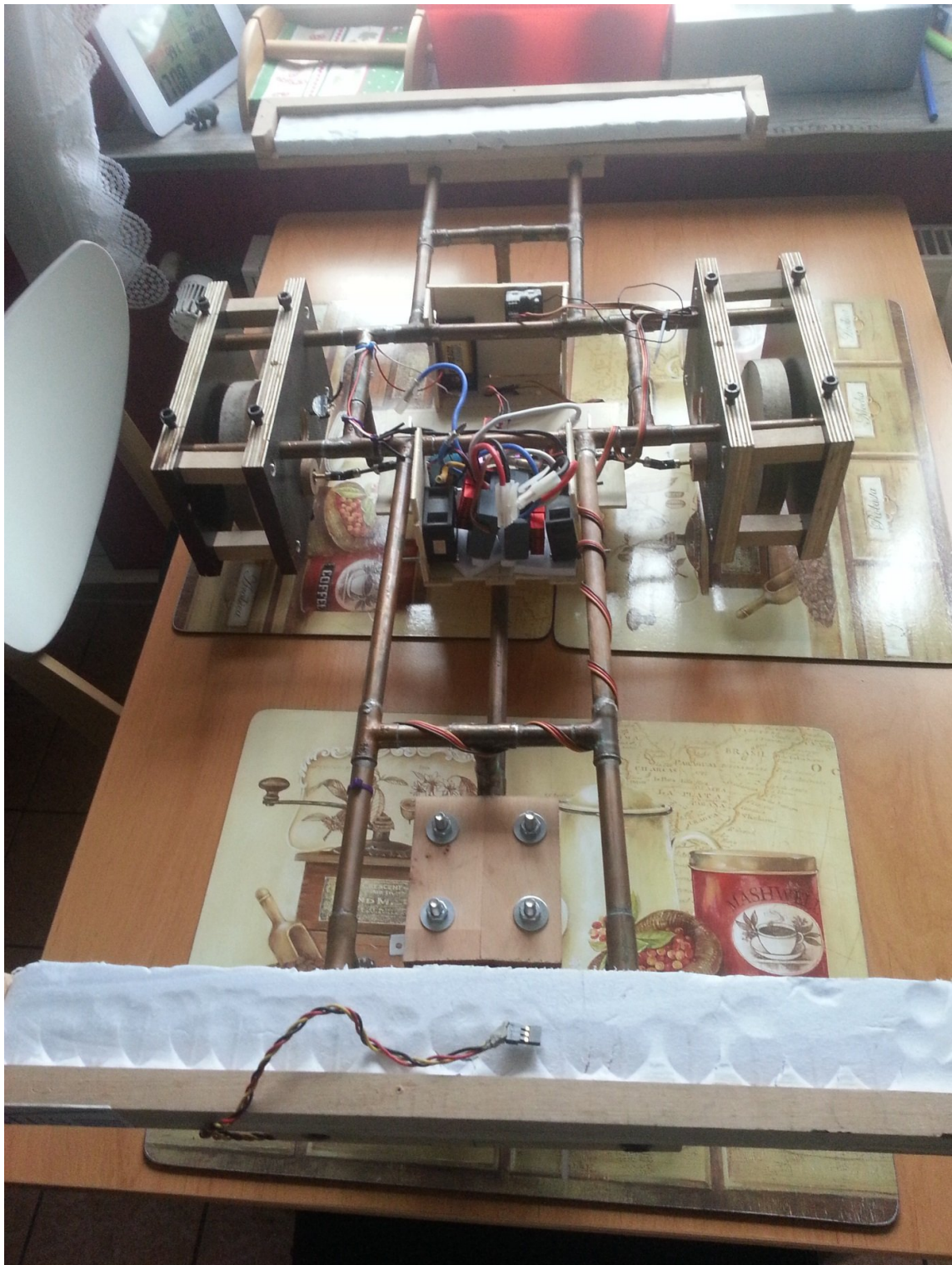
Hier sieht man die alte Technik, sowie den Platz den die Elektronik im Rahmen einnehmen wird.



Der Installationsaufbau des CarPC; Basierend auf einem Igel Thinclient.



Der Getriebekasten. Hier kommt nach den Osterferien das neue Getriebe rein. Die Räder werden durch Plastik/Gummi Räder ersetzt.



Gesamtaufnahme des Rahmens mit Solarpanelhalterungen.



Erste Einpassung des Mainboards. Es passt perfekt zwischen die Getriebekästen. Die Fahrregler werden zusammen mit dem Akku rechts vom Mainboard untergebracht. Die Motoren werden im Kasten unter dem Mainboard untergebracht. Sie werden mit Lüftern gekühlt.

Quellen:

<http://flyduino.net/bilder/produkte/normal/Flyduino-30A-ESC.jpg>

<http://de.wikipedia.org/wiki/Pulsweitenmodulation>

<http://de.wikipedia.org/wiki/Wlan#802.11+>

http://www.mikrocontroller.net/articles/Modellbauservo_Ansteuerung

<http://www.mikrocontroller.net/wikifiles/0/02/Servo.gif>

<http://www.usconverters.com/images/rs232-pinout.jpg>

Microsoft Visual Studio 2010 zur Erstellung der Klassen Diagramme

Alle anderen Diagramme wurden mit Microsoft PowerPoint 2010 erstellt.

Zentral Abitur Klassen (Angepasst durch Schüler der MES und Herr John):

-CSocket

-CServerSocket

-CSerial

Alle anderen Bilder wurden selber angefertigt.

Die Bilder zur Veranschaulichung des PWM-Signals wurden mit einem digitalen Speicher Oszilloskop gemacht.

Anhang:

Konzept/Anwendungsanforderungen

Klassendiagramm Server

Klasse CMotor

Klasse CError

Hauptprogramm Server und Client