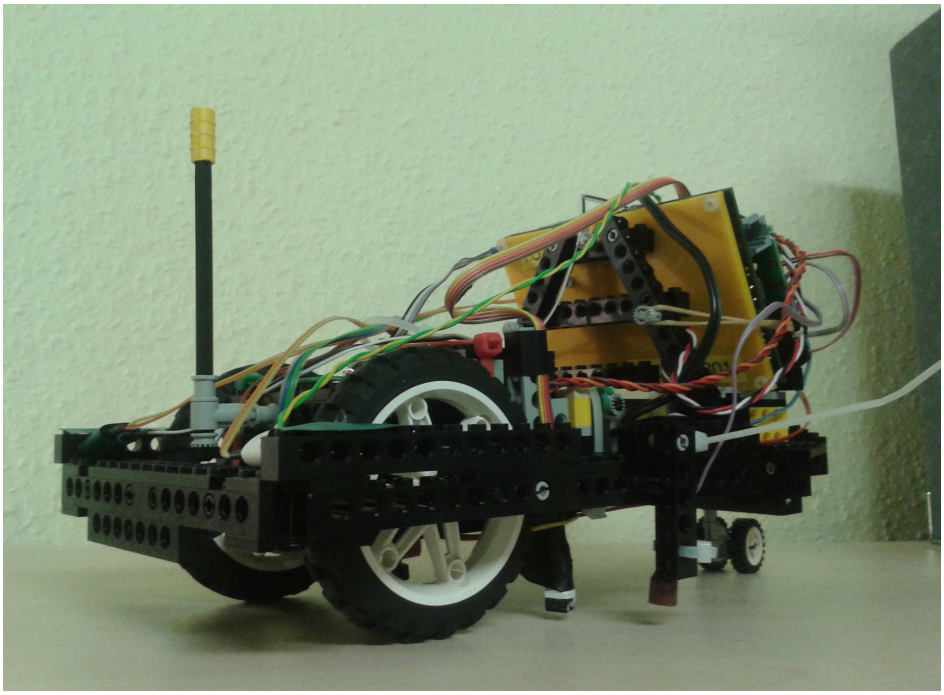
Autonome Mobile Systeme

FACHHOCHSCHULE
BRANDENBURG

Projektdokumentation
„Thunder“



vorgelegt von:

Dennis Schmidt
Sebastian Gräbitz

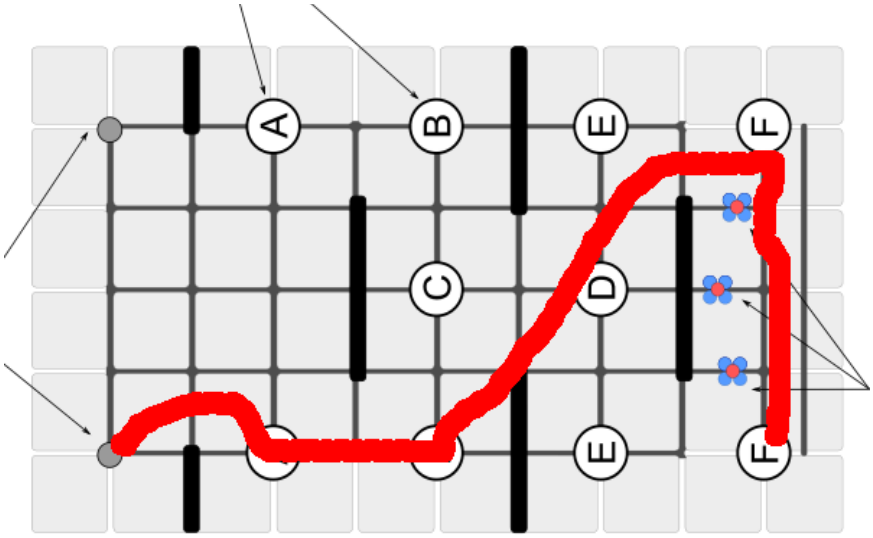
Inhalt:

1. Die Aufgabe

Die Aufgabe des Projektes bestand darin, mittels des in Norwegen entwickelten Kværner-Verfahren , Methan-Moleküle in Kohlenstoff und Wasserstoff zu zerlegen. Dadurch soll der sehr effizient gewonnene, reine Kohlenstoff, nutzbar gemacht werden. Um dies zu erreichen, wurden von den Teams autonome mobile Systeme entwickelt, die die Spaltung durchführen sollten.

2. Unsere Strategie

Zu Beginn des Projekts mussten wir uns also darüber einig werden, welcher Lösungsweg der Richtige für uns ist. Wollen wir um jeden Preis gewinnen oder einfach ein System liefern, dass die gestellten Aufgaben bewältigen kann?
Lange mussten wir dafür nicht überlegen. Wir fassten das ehrgeizige Ziel ins Auge, um jeden Preis gewinnen zu wollen.
Was sagt das über die anvisierte Strategie aus? Nach einigem Überlegen wurde uns klar, wir müssen nicht nur sehr schnell, sondern auch in hohem Maße zuverlässig sein. Aber zuerst zur Geschwindigkeit. Der Rest wird sich im Laufe des Entwicklungsprozessen schon von alleine ergeben. Rückblickend ist dies zugegebenermaßen eine optimistische Herangehensweise.



Der schnellste Weg wollte also gefunden werden. In der obigen Abbildung ist unsere Vorstellung dieses Weges rot markiert. Bei erleuchten des Startlichtes wollten wir in einem leichten Bogen in Richtung Wegpunkt A fahren, dann geradeaus zu Wegpunkt B. Ab hier sollte der Roboter diagonal über das Feld auf Wegpunkt E zu fahren. Parallel zur Außenwand des Feldes wollten wir Wegpunkt F erreichen und schließlich an den Molekülen vorbei fahren um die Aufgabe zu erfüllen.
Im Verlauf des Projekts mussten wir diesen Kurs jedoch leicht korrigieren, was uns höhere Zuverlässigkeit bescherte. Dazu später mehr.

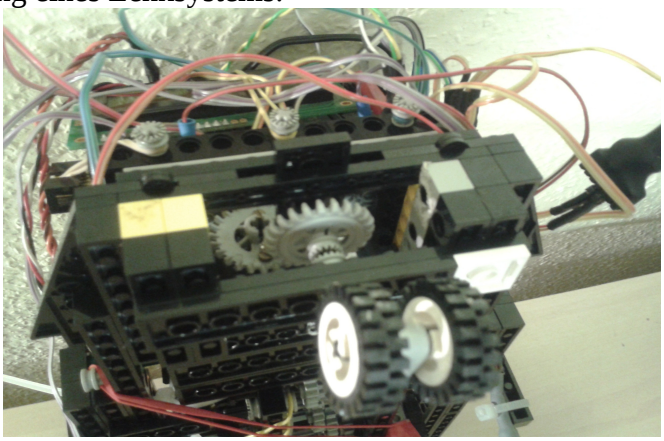
3. Die Evolution von „Thunder“

Der Name unseres Roboter ist „Thunder“ und er wird im weiteren Verlauf auch so genannt.

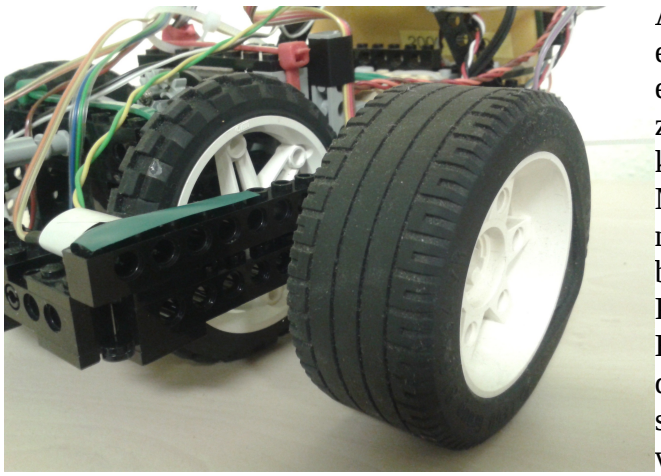
Jedoch: Am Anfang war Nichts.

Die erste große Herausforderung, vor der wir uns befanden war der Antrieb. Wie sollten wir Thunder das „Laufen“ beibringen?
Da wir immer auf Geschwindigkeit bedacht waren, bauten wir ein sehr, sehr schnelles Getriebe mit einer Übersetzung von 1:9. Mit diesem ersten Getriebe konnten wir enorm platzsparend bauen. Nachdem wir ein minimalistisches Chassis gebaut hatten und einen einzelnen Antriebsmotor verbauten, konnten wir relativ zügig feststellen, dass die Leistung des Motors nicht ausreichend war. Unbelastet konnten wir zwar hohe Umdrehungsgeschwindigkeiten erzielen, jedoch reichte das Eigengewicht des Roboters bereits aus, den Elektromotor an seine Grenzen zu bringen.

Fahren war mit diesem Getriebe nicht möglich. Also noch einmal von vorne anfangen. Wir schauten uns die zahlreichen Referenzmodelle im KI-Labor an und entwickelten danach ein Getriebe mit einer Übersetzung von 1:81 aus 4 Stufen. Wir konnten nun sehr gut die Kraft des Motors übertragen, waren allerdings auch deutlich langsamer. Trotzdem entschlossen wir uns dieses Getriebe zu verwenden.
Die nächste Herausforderung war die Entwicklung eines Lenksystems.



Es war für uns wichtig, geschmeidige Kurven fahren zu können. Denn wir befürchteten einen erheblichen Zeitverlust, wenn wir auf der Stelle wenden würden. Bis die Drehung vollzogen wäre, könnten wir ja nicht fahren. Das wollten wir auf gar keinen Fall, sondern wollten uns immer in einer Vorwärts-Bewegung befinden. So fassten wir den Entschluss, ein einzelnes Rad zum Lenken zu benutzen. Damit war es möglich sowohl große, als auch sehr kleine Radien zu fahren. Anhalten mussten wir dafür nicht.
Allerdings stellte sich bereits bei den ersten Tests heraus, dass die Reibung der angetriebenen Räder zu groß war um eine Kurvenfahrt zu vollführen. Ein erster Lösungsansatz bestand darin, die sehr breiten Vorderreifen durch dünnere Exemplare zu ersetzen. Gesagt getan.
Dies führte uns jedoch nicht zum Erfolg. Wir mussten weiterhin geradeaus fahren.

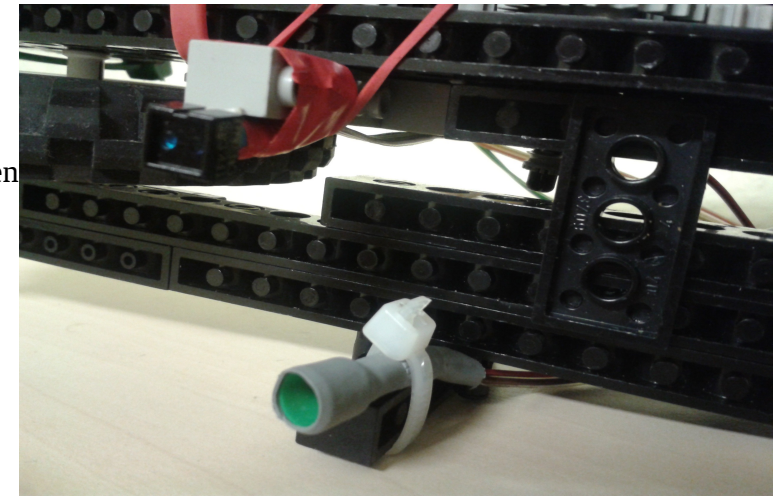


Aber uns gingen die Ideen noch nicht aus. Die einzige Lösung, die uns erfolgversprechend erschien, war der Einbau eines Differential zwischen den angetriebenen Rädern. Diese kleine Raffinesse entdeckten wir an einem Lego-Model aus unseren Kindertagen. Da wir uns noch in einem frühen Evolutionsstadium befanden, machte der Einbau keine großen Probleme. Endlich konnten wir erfolgreiche Ergebnisse vorweisen. Wir waren nicht nur in der Lage Kurven zu durchfahren, sondern stellten anhand des Getriebegeräusches fest, dass während der Kurvenfahrt so gut wie keine Leistungsverluste am Motor entstanden. Diese Tatsache wirkt sich ebenfalls begünstigend auf den Energieertrag der Akkus aus. Wir stellten uns daher die Frage: Wenn wir stetig nur wenig Leistung verbrauchen, sollten wir vielleicht eine komplett zeitgesteuerte Logik implementieren?
Wir versprachen uns dadurch einen erheblichen Gewichts- und Geschwindigkeitsvorteil gegenüber den anderen Teams. Nach einem Tag im KI-Labor konnten wir die Bewegungen von Thunder so weit optimieren, dass wir immer das Ziel am Ende des Spielfeldes erreichten. Dabei verfolgten wir genau den Weg, der in 2. beschrieben wurde.

Optimistisch und gut gelaunt starteten wir in den nächsten Labor-Tag.
Mit vollen Akkus ging es an die erste Testfahrt. Schnell machte sich Ernüchterung breit. Wir kamen nicht einmal annähernd ans Ziel. Mit den geladenen Akkus, waren wir sehr viel schneller als in den letzten Testfahrten. So scheiterten wir bereits am ersten Hindernis bei Wegpunkt B. Da hilft nur eins: optimieren.
Aber wie? Langsamer fahren? Schärfer lenken?
Wir wollten einen Mittelweg finden, der immer funktionieren sollte. Nach einem Tag Probieren, Verwerfen und wieder Probieren, gaben wir auf. Es fehlte uns eine gute Idee. Zum Ende des Tages

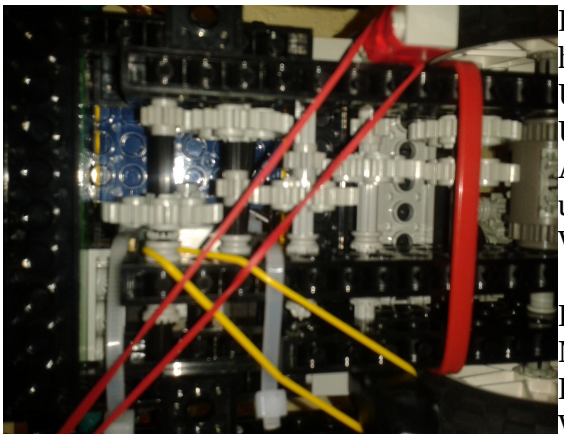
kam sie uns dann aber doch. Wir müssten einfach die Spannung der Akkus überwachen. Diese Möglichkeit besteht ja mit dem AKSEN-Board. Mit Zuversicht und Hoffnung machten wir das Licht im Labor aus.

Dass die eben erwähnte Idee auch nicht die Beste war, konnten wir dann ziemlich schnell feststellen. Das Board zeigt erst ab einer Spannung unter 5 Volt messbare Ergebnisse. Wir benötigen jedoch weit über 6 Volt Spannung für den reibungslosen Betrieb. So langsam sahen wir ein, dass es wohl ganz ohne Sensorik nicht gehen wird. Thunder stand kurz vor seiner nächsten Evolutionsstufe.



Die schwarzen Linien sollten uns nun also als Orientierung dienen. Wir spendierten unserem Roboter zwei Optokoppler und konnten somit schon eingeschränkt sehen. Ziel war es, mit den Sensoren, die ersten drei Linien zu zählen, dann entweder nach links oder nach rechts ab zu biegen und in die Diagonale überzugehen. In der Diagonalen weitere sechs Linien abwarten und erneut abbiegen um die Moleküle zu spalten. Siehe da, diese Strategie funktionierte bis Wegpunkt F sehr zuverlässig. Auch nach vielen Wiederholungen erreichten wir immer diesen Punkt.
Auf einmal jedoch wieder nicht. Dann wieder doch und oft wieder nicht. Frustration. Was ist denn nun schon wieder faul? Nach langem Überlegen mussten wir schließlich doch nachfragen. Wir bekamen den Tipp, dass die Optokoppler vielleicht nicht richtig messen. Um das zu kontrollieren nahmen wir unsere Handykameras, und machten so das infrarote Licht sichtbar. Beide leuchteten nicht. Es war uns ein Rätsel, wie Thunder überhaupt so oft an sein Ziel gelangen konnte.
Wir schalteten die Lichter also ein und alles funktionierte bestens. Diesen Tag konnten wir erfolgreich beenden.

Der nächste Labor-Termin brachte uns wieder zahlreiche Überraschungen.
Das Linienzählen hatten wir zwar in den Griff bekommen, aber wir waren immer noch sehr abhängig von unseren Akkus. Die Zeitsteuerung war ja immer noch ein tragender Teil des Verhaltens von Thunder. Wieder fingen wir an zu optimieren und das beste Gleichgewicht zwischen vollem und leerem Akku zu finden.
Dabei stießen wir immer wieder auf Unregelmäßigkeiten. Es verging einige Zeit bis wir heraus fanden, was uns das Leben schwer machte. Der zweite Teilabschnitt, die Diagonale, war die Antwort. Es kam immer wieder einmal vor, dass wir genau über den Schnittpunkt zweier schwarzer Linien gefahren sind. Somit erreichte Thunder die Anzahl von sechs Linien oft zu spät. Aus diesem Grund landeten wir regelmäßig an der Wand und Thunder konnte seinen Kurs nicht wieder finden.
Weitere störende Faktoren waren die Löcher zwischen Wegpunkt E und F. Für unseren Roboter eine sehr verwirrende Stelle, denn die Löcher wurden als Linien erkannt. Um diesen Gefahren entgegen zu wirken, schalteten wir nach der Diagonalen die Optokoppler aus und steuerten wieder zeitbasiert. Eine Methode die nicht ganz ungefährlich war, wie sich heraus stellte. Aufgrund der Länge unseres Akteurs, dem Laser zum Moleküle spalten, blieben wir gelegentlich an der letzten Hinderniswand zwischen Wegpunkt E und den Molekülen hängen, wodurch wir mit diesen kollidierten und Wasserstoff frei gesetzt wurde.
Zu allem Überfluss haben wir mittlerweile unseren Vorsprung an Geschwindigkeit an unsere Konkurrenz verloren.



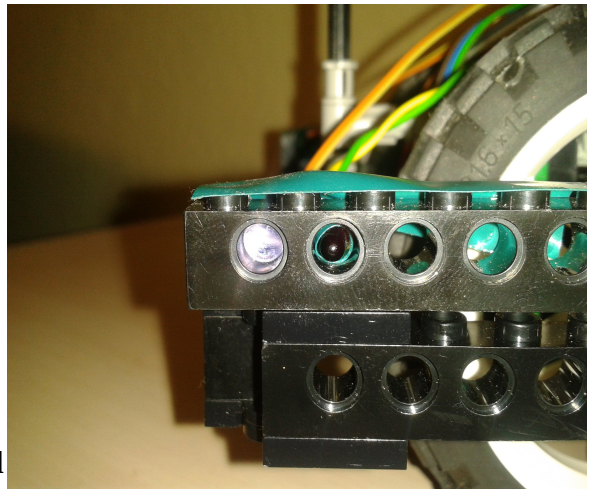
Diesen Verlust wollten wir auf gar keinen Fall hinnehmen. Wir änderten also noch einmal das Getriebe. Und zwar auf drei Stufen. Somit hatte Thunder eine Übersetzung von 1:27. Wir befürchteten, der einzige Antriebsmotor würde versagen, diese zweifel waren aber unberechtigt.
Wir waren nun wieder schnell genug.

Es blieb noch das Problem von Wegpunkt F zu den Molekülen zu finden.
Eine Strategie war schnell gefunden. Einfach von Wegpunkt E über Wegpunkt F hinaus gegen die Wand fahren, dann wenden und die Moleküle spalten. Aber wie sollten wir die Wand finden?
Wann müssen wir anhalten und wenden?
Wieder stand Thunder kurz vor einer evolutionären Veränderung.

Nach einiger Beratung mit Herrn Boersch entschieden wir uns für Abstandsmessung mit einem IR-Sender und einem IR-Empfänger an der Vorderseite.
Wir machten uns mit der Funktionsweise dieses Sensors vertraut und erweiterten Thunder's Fähigkeiten. Unser Roboter hatte nun sein erstes Auge. Leider schien es blind zu sein, denn sobald die Wand erreicht wurde, stoppte Thunder nicht wie erwartet. Wir erinnerten uns an die Probleme mit den Optokopplern und holten unsere Telefone heraus. Die Diode war falsch an das Board gesteckt. Spätestens zu diesem Zeitpunkt wurden die Telefone unsere besten Freunde und Thunder konnte sehen.
Also wieder zu unserem Plan. An Wegpunkt E und F vorbei und möglichst schnell gegen die Außenwand fahren, so dass der Roboter idealerweise orthogonal zur Wand steht.
Einige Versuche später stellten wir fest, der Getriebeumbau hat zur Folge, dass Thunder nicht genug Kraft besitzt, sich an der Wand auszurichten. Wir kamen immer wieder in einem anderen Winkel zum Stehen. Das brachte uns also keinerlei Vorteile.

Ein weiterer Labor-Tag ging zu Ende.

Da wir gute Erfahrungen mit der Abstandsmessung in Hinblick auf Genauigkeit und Geschwindigkeit gemacht haben, entschieden wir vier weitere dieser Sensoren zu verwenden, und zwar links und rechts, jeweils vorn und hinten. Weil wir nur vier LED-Steckplätze zur Verfügung hatten, jedoch fünf benötigen, mussten wir einen Verteiler verbauen. Für Mensch und Maschine nicht ganz ungefährlich, wie sich noch zeigen sollte.



Nachdem alle Sensoren verbaut waren, haben wir, schon fast paranoid, die Funktionsweise der LEDs überprüft. Alles schien einwandfrei zu funktionieren.
Also bestimmten wir die Grenzwerte für die Abstände zu den Wänden. Auf der rechten Seite stellten wir fest, dass beide Sensoren nahezu die gleichen Werte liefern, wenn der Roboter parallel zu zu einer Wand steht.
Links war das gar nicht der Fall. Bei gleichem Abstand zur Wand war ein Messwert fast doppelt so groß wie der Andere. Wir tauschten also einen IR-Sender und einen IR-Empfänger aus und erhielten danach auch gleiche Messwerte. Offenbar senden einige LEDs mehr Licht aus als andere.
Wir stellten also die Grenzwerte ein und testeten drauf los.

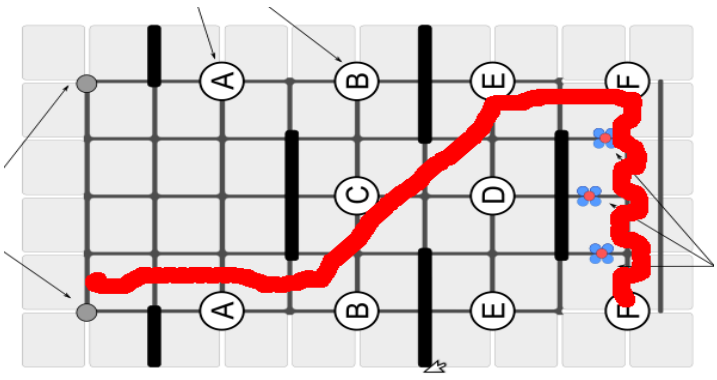
Thunder sollte sich nach der Diagonalen, an einer Wand links oder rechts, parallel mit den seitlichen Sensoren ausrichten, auf die nächste Wand zu fahren bis ein Schwellwert erreicht ist, und sich dann mit den seitlichen Sensoren (rechts oder links) an dieser Wand ausrichten. Danach sollte der Roboter der Wand folgen, um die Moleküle zu spalten und schließlich kurz vor der nächsten Wand anhalten und sich beenden. Soweit die Theorie.
Doch die Praxis sah überraschend anders aus.
Thunder verfiel immer häufiger in undefinierbare Zustände. Fuhr, hielt an, lenkte links anstatt rechts und schaltete sich immer häufiger ab. Programmschleifen wurden unkontrolliert ausgeführt, oder auch nicht. Kurz bevor wir den Tag demotiviert und frustriert beendeten, haben wir starke Hitzeentwicklung am AKSEN-Board wahrgenommen. Ein Test mit den Fingern bestätigte diese Wahrnehmung auf schmerzliche Weise. Messungen zeigten einen erhöhten Stromverbrauch jenseits der 2 Ampere-Marke. Das Board schaltete sich also aus Sicherheitsgründen ab um nicht zerstört zu werden. Nach einigem Suchen fanden wir den Übeltäter. Schuld war ein Kurzschluss im Verteiler für die LEDs.
Schnell war das defekte Bauteil getauscht. Aber wie war das jetzt mit den LEDs? Stecken die noch richtig herum auf dem Board? Kontrollieren. Als wir sicher waren, dass alle Elektronen den richtigen Weg nehmen, konnten wir unsere Entwicklung fortsetzen.
Die nächste Herausforderung stellte die Ecke bei Wegpunkt E dar. Je nach dem, in welchem Winkel Thunder auf die Ecke zu fuhr, verhielt sich der Roboter anders. Dass wir überhaupt in verschiedenen Winkeln an der Ecke ankommen, ist der Tatsache geschuldet, dass wir nach dem Start einen kleinen Bogen (wie unter 2. beschrieben) fahren. Der Roboter müsste exakter ausgerichtet werden können. Wir stellten also fest, dass es in beschriebener Ecke zu unvorhergesehenen Reflexionen gekommen war, und so die Messergebnisse verfälscht wurden.

Thunder kann zu gut sehen.

Daher schränkten wir sein Sichtfeld ein. Bevor wir nicht bis zu einem vorher bestimmten Grenzwert an eine Wand herangefahren sind, durfte Thunder keine Aktion ausführen. Das führte zu einem sehr zuverlässigen Verhalten. Bei jeder Fahrt erreichten wir das gewünschte Ziel.
So konnten wir uns auf den nächsten Tag im Labor freuen.

Nachdem wir nun alle Sensoren angeschlossen hatten, standen wir vor dem nächsten Rätsel. Mitten auf dem Spielfeld verlangsamte sich Thunder, quälte sich einige Zentimeter, um dann wieder die volle Leistung zu entfalten. Sofort gingen unsere Köpfe unter den Tisch. Hatte dort jemand einen Magneten vergessen? Dem war natürlich nicht so. Was war dann die Erklärung?
Für uns war das ganz klar. Durch die Tests mit dem Kurzschluss und der Überhitzung des Boards, war es offenbar defekt. Was nun? Das AKSEN-Board tauschen und erneut alle Sensoren auf Funktion testen? Nein! Wir haben den wahren Grund für dieses Verhalten dann doch noch heraus gefunden. Mittlerweile verwendeten wir ja fünf LEDs zur selben Zeit. Dafür reicht allerdings die Akkuleistung nicht aus. Also können wir nur so viele LEDs einschalten, wie wirklich benötigt werden. Zudem schalteten wir die Dioden in sehr kurzen Intervallen immer wieder ein und aus. Somit war auch dieses Problem gelöst.

Was nun noch fehlte war die Fahrt entlang der Moleküle um diese endlich erfolgreich zu spalten. Da wir unsere Messung in kleinen Intervallen vollführten, konnten wir auch nur in diesen Intervallen unseren Kurs korrigieren. Daraus resultierte eine geschlängelte Linie. Jedoch konnten wir unseren Kurs relativ gut stabil halten und um einen kleinen Toleranzbereich einen



geradlinigen Kurs verfolgen. Leider stellten wir fest, dass der Abstand zur hinteren Wand immer unterschiedlich war. So kam es vor, dass unser Akteur die Moleküle manchmal nicht erreichte. Erste Überlegungen den Akteur zu verlängern, führten zu keinem Erfolg. Der eingesetzte Servomotor konnte das Gewicht nicht mehr halten. Die zweite Tatsache, die zum Misserfolg führte war, dass der Akteur unter Umständen zu lang gewesen wäre. Uns blieb also nur übrig, den Abstand zur Wand zu regulieren. Auch das gelang uns, nach einigen Versuchen, recht gut. Wir fuhren so gut wie immer die selbe Linie.
Ein Tag ging erfolgreich zu Ende.

Am letzten Tag wollten wir lediglich kleine Unregelmäßigkeiten beseitigen. Die erste Kurvenfahrt nach dem Start bereitete uns Kopfzerbrechen. Wir mussten Thunder haargenau und exakt ausrichten, um den anvisierten Kurs zu treffen. Dies gelang in den seltensten Fällen. Ein letztes Mal bauten wir unseren Roboter um. Durch die zahlreichen Anbauten für die Sensorik waren wir nun ausreichend breit. Die Lichtsteuerung zum Start des Roboters konnte somit nach außen verlegt werden. Wir konnten ab jetzt direkt nach dem Start geradeaus fahren. Die Ausrichtung erfolgte an den Markierungen auf dem Spielfeld. Eine sehr viel zuverlässigere Methode.
Ein paar kleine Veränderungen am Programmcode wollten noch ausprobiert werden.
Doch schon lauerte der nächste Rückschlag.
Thunder verhielt sich nicht einmal ansatzweise wie erwartet. Motore wurden unkontrolliert angesteuert und Lampen wild an- und ausgeschaltet. Auch intensive Codeanalyse brachte uns auf keine Lösung. Alles war wie bisher, nur der ein oder andere Wert wurde geringfügig verändert. Verzweifelt ob der knappen Zeit, wendeten wir uns abermals an Herrn Boersch. Doch auch mit vereinter Kraft wurden wir nicht fündig. Auf einmal jedoch, fiel folgender Satz: „Bleibt noch die Möglichkeit, dass nicht im Flash steht, was wir hier sehen.“.
Natürlich! Während des ganzen Projekts hatte niemand andere Einstellungen vorgenommen. Wenn wir den Programmflasher aufriefen, war immer der richtige Pfad zum Programmcode von Thunder eingestellt. Diesmal jedoch nicht. Danke Herr Boersch, für den heißen Tip.
Schnell noch den richtigen Code geflasht und siehe da: Thunder arbeitet zuverlässig und schnell.

Die oberste Evolutionsstufe ist erklommen und wir konnten dem Wettstreit zuversichtlich entgegen sehen.

4. Der Quellcode

Ebenso wie die technische Entwicklung des Roboters evolutionierte sich auch der Quellcode. Den Anfang machte eine primitive Zeitsteuerung. Mit dem „sleep“ Befehl steuerten wir so den Radius von Kurven und die Dauer von Geradeausfahrten. Das sah nach den ersten Versuchen sehr viel versprechend aus und wir untergliederten den Quellcode in verschiedene Checkpoints. Der Hintergrund der Checkpoints war dieser, dass wir die einzelnen Etappen auf dem Weg zu den Molekülen abspeichern konnten, in dem wir einfache Variablen auf 0 bzw. 1 setzten und später anhand dessen eine Bewältigung von nicht vorhersehbaren Hindernissen implementieren wollten. Es war angedacht beim Eintritt von unerwarteten Fehlern oder Hindernissen die gefahrene Strecke so auswertbar zu machen, dass wir den selben Weg den Thunder schon gefahren war wieder rückwärts oder umgedreht zurück zum Start fahren zu können. Um eine Übersicht aller angeschlossenen Sensoren und Motore zu haben, beschlossen wir als erstes einen abgeschlossenen Bereich im Quellcode zu definieren, so dass wir als Kommentare die einzelnen AKSEN-Board Belegungen immer parat haben. Da die Zeitsteuerung nicht von Dauer war, wurde zügig und unkompliziert eine Methode implementiert die Thunder mit Hilfe von 2 Optokopplern eine übergebene Anzahl von Linien überfahren ließ. Diese Funktion war in ihrer ersten Version leider wenig durchdacht, da Thundern nun in der Lage war, durch die schnelle Abtastrate, ein und dieselbe Linie mehrmals zu zählen. Schnell war ein zweiter Parameter definiert, der uns die Möglichkeit gab eine feste Zahl von Millisekunden nach jeder Messung zu warten. Wir ließen Thunder eine Strecke

von drei Metern fahren und rechneten uns seine Geschwindigkeit aus. Diesen Wert setzten wir ins Verhältnis zu der Breite einer Linie und wussten nun, dass wir circa 50 Millisekunden zwischen jeder Messung pausieren müssen, um keine Linie doppelt zu zählen, aber auch keine zu übersehen. Ein paar Fahrten bewiesen die Zuverlässigkeit der Methode auf geraden Strecken. Wir teilten die Methode im nächsten Atemzug, so dass wir für den Optokoppler auf jeder Seite eine eigene erhielten. Das war notwendig, da wir zum Abbiegen auf die Gerade auf der Kurvenaußenseite mit dem Optokoppler eine Linie finden wollten, daher nur mit einem von beiden messen brauchten, um auch diese Kurve nicht mehr zeitgesteuert fahren zu müssen.
Der Start verlief seit dem Beginn des Programms stets gleich und zuverlässig. Wir entschieden uns einmalig bei Initialisierung mit dem Lichtintensitätssensor das Umgebungslicht zu messen und abzuspeichern. Danach wurde eine Schleife durchlaufen, welche permanent den aktuellen Wert misst und ihn mit dem Initialwert abzüglich einer von uns gegebenen Toleranz vergleicht. Demnach war eine gewisse Ausfallsicherheit gegeben, falls das Umgebungslicht stark schwanken sollte. Die Startposition wurde anhand des Optokopplers ermittelt, welcher beim Start auf einer schwarzen Linie stand. So wusste Thunder immer welche Route er zu nehmen hatte und startete zuverlässig.
Da das diagonale Fahren sehr fehleranfällig war und der Roboter nur in manchen Fällen unseren Erwartungen nach korrekt zählte, entschlossen wir uns die angesprochene Sensorik mit Hilfe von Infrarot-Sendern und Empfängern zu verwenden. Die Idee dahinter war mit Hilfe der erfassten Werte einen Abstand zu repräsentieren und somit genauer auf dem Spielfeld agieren zu können. Eine weitere Funktion musste her. Das Prinzip war dank guter Erklärung schnell verstanden und es ging an die Umsetzung. Als erstes mussten mehrere Variablen angelegt werden. Eine die das Infrarotlicht der Umgebung speicherte. Bevor wir die zweite Variable, den Messwert des Infrarotempfängers entgegen nahmen, wurde die Infrarotdiode angeschaltet, welche neben dem Empfänger saß und in dieselbe Richtung zeigte. Nun wurde ein weiteres Mal gemessen und es ergab sich, umso näher der Roboter der Wand kam, eine höhere Zahl, da der Lichtkegel der Diode intensiver wurde. Der zweite Messwert spiegelt also den Anteil des Infrarotlichtes in der Umgebung und das ausgesandte Licht der Diode wieder. Die Messwerte wurden gleicher umso Höher der Anteil des Infrarotlichtes war. Demnach musste nur noch in einer Abfrage beide Werte verglichen werden und in dem Fall, dass der Gesamtwert kleiner als der erste Wert ist, der Infrarot-Referenzwert, nämlich die Differenz nur aus dem Umgebungslicht und des Gesamtwertes, in einer globalen Variable abgespeichert werden. Beim Aufruf der Methode wurde also dieser Infrarot-Referenzwert erzeugt welcher wiederum in kurzen Abständen aktualisiert werden konnte und somit ein genauer Abstand zu Wänden nachvollziehbar bereitgestellt wurde. Nachdem noch pro Seite 2 weitere Infrarotmessstationen hinzukamen, wurden alle „Augen“ innerhalb einer Methode abgefragt. Im Zuge dessen litt die Akkuleistung sehr und wir reduzierten die Abfrage der Werte auf die jeweilige Seite, auf der wir eine Wand suchten. Die Front, Links- und Rechtssensorik erhielten also jeweils ihre eigene Funktion zum Generieren der Infrarotreferenzwerte. Nun war auch dieses Problem gelöst und wir konnten nach dem Abbiegen auf die Diagonale die auf uns zukommende Wand suchen und auch finden.
Nun war die letzte Herausforderung, die Orientierung parallel zu einer Wand. Da wir jeweils hinten und vorne auf einer Seite eine Messstation hatten waren wir in der Lage die unterschiedlichen Werte beider Messungen gegeneinander zu Vergleichen und demnach den Roboter näher an die Wand zu steuern oder umgekehrt. Es entstand zwar eine geschlängelte Linie, aber sie war sauber und gleichmäßig.
Der nächste Schritt bestand darin einen gewissen Abstand zur Wand einhalten zu können. Da wir schon die Funktionalität des Vergleiches des hinteren und vorderen Messwertes implementiert hatten, brauchten wir nur noch den Abstand als Bereich definieren und mit in die Abfrage zu integrieren.
Im Zuge der Umstrukturierung zu den Infrarotsensoren wurden auch die einzelnen Checkpoints im Programm entfernt, da wir nun eine wesentlich höhere Zuverlässigkeit erzielten. Ebenso wurde nun gerade gestartet. Da wir keine Kurve mehr zum Anfang fahren brauchten war es auch nicht mehr notwendig Thunder mit einem Optokoppler auf eine schwarze Linie zu stellen. Die Richtung ergab

sich nun daraus, welcher der beiden Lichtsensoren den Impuls bekam. Dementsprechend wurde eine Variable gesetzt und im Verlauf des Programms war stets klar, welche Route zu fahren ist.
Der Code des Roboters ist nicht zu hundert Prozent ausfallsicher. In der letzten Kurve zu den Molekülen kann es z.B. passieren, dass er sehr ungünstig mit beiden Messstationen einer Seite in die Ecke leuchtet und sich ein Wertepaar ergibt welches ihn dazu veranlasst geradeaus zu fahren anstatt die Kurve zu beenden. Da die Terminierung des Programms mit einem Abstand von der Front des Roboters zu einer Wand geschieht, wurde sie unter oben genannten Umständen zu früh ausgelöst und der Roboter blieb in der Ecke stehen.
Ansonsten wurde der Quellcode an vielen Stellen bereits so optimiert (Beispiel: Einberechnung des Umgebungslichtes, Infrarotanteil des Umgebungslichtes,…), dass es zu wenig Ausfällen während der Fahrt kommt.

5. Fazit

Trotz vieler Rückschläge und frustrierender Minuten im Labor, haben wir doch im Verlauf des Projektes, viel lernen dürfen.
Uns fiel auf, dass wir zwar viel erreichen wollten, jedoch erst einmal klein anfangen mussten um Fortschritte zu machen. Wir stellten fest, dass sowohl Erfolg, als auch Misserfolg, sehr nahe bei einander liegen und Fehler in allen Komponenten des Systems vorkommen können. Die Herausforderung bestand stets darin, an jede Kleinigkeit zu denken und diese auch in Betracht zu ziehen.
Aus einer vermeindlich einfachen Aufgabe, haben sich komplexe Probleme ergeben, die es zu lösen galt. Dabei haben wir viel Spaß gehabt und einiges gelernt.

Vielen Dank dafür.

```c
//Autor: K.-U. Mrkor, Sebastian Graebitz, Dennis Schmidt

//Standard-Include-Files
#include <stdio.h>
#include <regc515c.h>

//Diese Include-Datei macht alle Funktionen der
//AkSen-Bibliothek bekannt.
//Unbedingt einbinden!
#include <stub.h>

unsigned char ir_f = 0;
unsigned char ir_fr = 0;
unsigned char ir_hr = 0;
unsigned char ir_fl = 0;
unsigned char ir_hl = 0;

void linien_ueberfahren_r(int anzahl_linien, int mSec)
{
    int linien_counter_r = 0;
    int wert_rechts = 0;

    while(linien_counter_r < anzahl_linien)
    {
        wert_rechts = analog(0);

        if (wert_rechts > 100)
        {
            linien_counter_r++;
            sleep(mSec);
        }
    }
}


void linien_ueberfahren_l(int anzahl_linien, int mSec)
{
    int linien_counter_l = 0;
    int wert_rechts = 0;

    while(linien_counter_l < anzahl_linien)
    {
        wert_rechts = analog(2);

        if (wert_rechts > 100)
        {
            linien_counter_l++;
            sleep(mSec);
        }
    }
}


void ir_init()
{
//analog(1) = Infrarotempfänger VORNE
//analog(4) = Infrarotempfänger VORNE_LINKS
//analog(3) = Infrarotempfänger VORNE_RECHTS
//analog(6) = Infrarotempfänger HINTEN_LINKS
//analog(5) = Infrarotempfänger HINTEN_RECHTS
//led 0     = LED des Infrarotsenders VORNE
//led 1     = LED des Infrarotsenders VORNE_RECHTS
//led 2     = LED der Infrarotsender VORNE_LINKS,HINTEN_LINKS
//led 3     = LED der Infrarotsender HINTEN_RECHTS

    unsigned char r = 0;
    unsigned char solar_f = 0;
    unsigned char solar_fr = 0;
    unsigned char solar_hr = 0;
    unsigned char solar_fl = 0;
    unsigned char solar_hl = 0;
    unsigned char ir_gesamt_f = 0;
    unsigned char ir_gesamt_fr = 0;
    unsigned char ir_gesamt_hr = 0;
```

```
    unsigned char ir_gesamt_fl = 0;
    unsigned char ir_gesamt_hl = 0;

    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);
    solar_f = analog(1);
    solar_fr = analog(3);
    solar_hr = analog(5);
    solar_fl = analog(4);
    solar_hl = analog(6);
    led(0,1);
    led(1,1);
    led(2,1);
    led(3,1);
    sleep(10);
    ir_gesamt_f = analog(1);
    ir_gesamt_fr = analog(3);
    ir_gesamt_hr = analog(5);
    ir_gesamt_fl = analog(4);
    ir_gesamt_hl = analog(6);

    if (ir_gesamt_f < solar_f)
    {
        ir_f = solar_f - ir_gesamt_f;
    }
    else
    {
        ir_f = 0;
    }

    if (ir_gesamt_fr < solar_fr)
    {
        ir_fr = solar_fr - ir_gesamt_fr;
    }
    else
    {
        ir_fr = 0;
    }

    if (ir_gesamt_hr < solar_hr)
    {
        ir_hr = solar_hr - ir_gesamt_hr;
    }
    else
    {
        ir_hr = 0;
    }

    if (ir_gesamt_fl < solar_fl)
    {
        ir_fl = solar_fl - ir_gesamt_fl;
    }
    else
    {
        ir_fl = 0;
    }

    if (ir_gesamt_hl < solar_hl)
    {
        ir_hl = solar_hl - ir_gesamt_hl;
    }
    else
    {
        ir_hl = 0;
    }

    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);
}
```

```
void ir_init_front()
{
//analog(1) = Infrarotempfänger VORNE
//analog(4) = Infrarotempfänger VORNE_LINKS
//analog(3) = Infrarotempfänger VORNE_RECHTS
//analog(6) = Infrarotempfänger HINTEN_LINKS
//analog(5) = Infrarotempfänger HINTEN_RECHTS
//led 0     = LED des Infrarotsenders VORNE
//led 1     = LED des Infrarotsenders VORNE_RECHTS
//led 2     = LED der Infrarotsender VORNE_LINKS,HINTEN_LINKS
//led 3     = LED der Infrarotsender HINTEN_RECHTS


    unsigned char r = 0;
    unsigned char solar_f = 0;
    unsigned char ir_gesamt_f = 0;

    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);
    solar_f = analog(1);

    led(0,1);

    sleep(10);
    ir_gesamt_f = analog(1);

    if (ir_gesamt_f < solar_f)
    {
        ir_f = solar_f - ir_gesamt_f;
    }
    else
    {
        ir_f = 0;
    }

    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);
}

void ir_init_rechts()
{
//analog(1) = Infrarotempfänger VORNE
//analog(4) = Infrarotempfänger VORNE_LINKS
//analog(3) = Infrarotempfänger VORNE_RECHTS
//analog(6) = Infrarotempfänger HINTEN_LINKS
//analog(5) = Infrarotempfänger HINTEN_RECHTS
//led 0     = LED des Infrarotsenders VORNE
//led 1     = LED des Infrarotsenders VORNE_RECHTS
//led 2     = LED der Infrarotsender VORNE_LINKS,HINTEN_LINKS
//led 3     = LED der Infrarotsender HINTEN_RECHTS


    unsigned char solar_fr = 0;
    unsigned char solar_hr = 0;
    unsigned char ir_gesamt_fr = 0;
    unsigned char ir_gesamt_hr = 0;


    led(0,0);
    led(2,0);
    led(1,0);
    led(3,0);
    solar_fr = analog(3);
    solar_hr = analog(5);
    led(0,1);
    led(1,1);
    led(2,1);
    led(3,1);
    sleep(10);
    ir_gesamt_fr = analog(3);
```

```
    ir_gesamt_hr = analog(5);

    if (ir_gesamt_fr < solar_fr)
    {
        ir_fr = solar_fr - ir_gesamt_fr;
    }
    else
    {
        ir_fr = 0;
    }

    if (ir_gesamt_hr < solar_hr)
    {
        ir_hr = solar_hr - ir_gesamt_hr;
    }
    else
    {
        ir_hr = 0;
    }

    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);
}

void ir_init_links()
{
//analog(1) = Infrarotempfänger VORNE
//analog(4) = Infrarotempfänger VORNE_LINKS
//analog(3) = Infrarotempfänger VORNE_RECHTS
//analog(6) = Infrarotempfänger HINTEN_LINKS
//analog(5) = Infrarotempfänger HINTEN_RECHTS
//led 0     = LED des Infrarotsenders VORNE
//led 1     = LED des Infrarotsenders VORNE_RECHTS
//led 2     = LED der Infrarotsender VORNE_LINKS,HINTEN_LINKS
//led 3     = LED der Infrarotsender HINTEN_RECHTS

    unsigned char solar_fl = 0;
    unsigned char solar_hl = 0;
    unsigned char ir_gesamt_fl = 0;
    unsigned char ir_gesamt_hl = 0;

    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);
    solar_fl = analog(4);
    solar_hl = analog(6);
    led(2,1);

    sleep(10);

    ir_gesamt_fl = analog(4);
    ir_gesamt_hl = analog(6);

    if (ir_gesamt_fl < solar_fl)
    {
        ir_fl = solar_fl - ir_gesamt_fl;
    }
    else
    {
        ir_fl = 0;
    }

    if (ir_gesamt_hl < solar_hl)
    {
        ir_hl = solar_hl - ir_gesamt_hl;
    }
    else
    {
        ir_hl = 0;
    }
```

```
    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);
}


void wand_suchen(unsigned char mode, unsigned char richtung)
{
    unsigned char wand = 0;

    while (wand == 0)
    {
        ir_init_front();

        if (mode == 0)
        {
            if (ir_f > 3)
            {
                wand = 1;
            }
        }
        else if (mode == 1)
        {
            if (ir_f > 3)
            {
                wand = 1;
            }
        }
    }

    if (richtung == 1) //LINKS
    {
        //scharf nach links lenken
        servo_arc(0, 115);
        ir_init_rechts();

        if (mode == 0)
        {
            while (ir_fr <= 10)
            {
                ir_init_rechts();
            }

            while (ir_fr > ir_hr)
            {
                ir_init_rechts();
                //scharf nach links lenken
                servo_arc(0, 115);
            }
        }
        else if (mode == 1)
        {
            while (ir_fr < ir_hr)
            {
                ir_init_rechts();
                //scharf nach links lenken
                servo_arc(0, 115);
            }
        }
        if (mode == 0)
        {
            //geradeaus
            servo_arc(0, 77);
        }
        else
        {
            //scharf nach rechts lenken
            servo_arc(0, 56);
        }
    }
    else if (richtung == 0) //RECHTS
    {
        //scharf nach rechts lenken
```

```
        servo_arc(0, 39);
        ir_init_links();

        if (mode == 0)
        {
            while (ir_fl <= 14)
            {
                ir_init_links();
            }

            while (ir_fl > ir_hl)
            {
                ir_init_links();
                //scharf nach rechts lenken
                servo_arc(0, 39);
            }
        }
        else if (mode == 1)
        {
            while (ir_fl < ir_hl)
            {
                ir_init_links();
                //scharf nach rechts lenken
                servo_arc(0, 39);
            }
        }
        if (mode == 0)
        {
            //geradeaus
            servo_arc(0, 77);
        }
        else
        {
            //scharf nach links lenken
            servo_arc(0, 98);
        }

    }
}

void ausrichtenWandRechts()
{
    unsigned char wand = 0;

    while (wand == 0)
    {

        ir_init_rechts();

        while ((ir_fr > ir_hr) && (ir_fr > 14))
        {
            ir_init_rechts();
            //scharf nach links lenken
            servo_arc(0, 98);
        }

        //Akteur betaetigen, um Atome zu spalten
        servo_arc(2, 66);

        while (wand == 0)
        {
            ir_init_rechts();

            if ((ir_fr < 14) && (ir_fr > 10))
            {

            }
            else if (ir_fr < ir_hr)
            {
                //scharf nach rechts lenken
                servo_arc(0, 56);
            }
            else if(ir_fr > 14)
            {
```

```
                //scharf nach links lenken
                servo_arc(0, 98);
            }
            else if (ir_fr < 10)
            {
                //scharf nach rechts lenken
                servo_arc(0, 56);
            }

            ir_init_front();

            if (ir_f > 10)
            {
                wand = 1;
            }
        }
    }
}

void ausrichtenWandLinks()
{
    unsigned char wand = 0;

    while (wand == 0)
    {

        ir_init_links();

        while ((ir_fl > ir_hl) && (ir_fl > 18))
        {
            ir_init_links();
            //scharf nach rechts lenken
            servo_arc(0, 56);
        }

        //Akteur betaetigen, um Atome zu spalten
        servo_arc(2, 115);

        while (wand == 0)
        {
            ir_init_links();

            if ((ir_fl < 18) && (ir_fl > 14))
            {

            }
            else if (ir_fl < ir_hl)
            {
                //scharf nach links lenken
                servo_arc(0, 98);
            }
            else if(ir_fl > 18)
            {
                //scharf nach rechts lenken
                servo_arc(0, 56);
            }
            else if (ir_fl < 14)
            {
                //scharf nach links lenken
                servo_arc(0, 98);
            }

            ir_init_front();

            if (ir_f > 10)
            {
                wand = 1;
            }
        }
    }
}


//Hauptprogrammroutine
```

```
void AksenMain(void)
{

//analog(8) = Lichsensor fuer den Start

//analog(0) = Optokoppler zum Zaehlen der Linien rechts
//analog(2) = Optokoppler zum Zaehlen der Linien links

//analog(1) = Infrarotempfänger VORNE
//analog(4) = Infrarotempfänger VORNE_LINKS
//analog(3) = Infrarotempfänger VORNE_RECHTS
//analog(6) = Infrarotempfänger HINTEN_LINKS
//analog(5) = Infrarotempfänger HINTEN_RECHTS
//led 0     = LED des Infrarotsenders VORNE analog(4)
//led 1     = LED der Infrarotsender VORNE_LINKS,VORNE_RECHTS,HINTEN_LINKS,HINTEN_RECHTS
    analog(3)

//servo_arc 0 = Servomotor zum Lenken
//servo_arc 2 = Servomotor fuer den Akteur
//motor_pwm 0 = Motor fur den Antrieb


//Variable ist 1, solange Thunder faehrt
unsigned char fahren = 0;
//Varaible ist 1, wenn Thunder links startet
unsigned char links = 0;
//Variable ist dann 1, wenn Thunder Aufgaben erfuellt hat und das Programmende erreicht ist
unsigned char ende = 0;
//Checkpoints in der Programmlogi, die auf 1 gesetzt werden, sobald sie erreicht wurden
unsigned char phase = 0;
//gibt an welcher Optokoppler auf der Diagonalen mehr Linien gezaehlt hat links = 1, rechts
     = 0
unsigned char links_diag = 0;
//Variablen fuer den Startschuss und die Richtung
int wert_licht_l = analog(8);
int wert_licht_r = analog(14);

    led(0,0);
    led(1,0);
    led(2,0);
    led(3,0);


/* Diagnose der linken Sensorenwerte
led(0,1);
led(1,1);
led(2,1);
led(3,1);

while (1)
{
    ir_init();
    lcd_cls;
    lcd_setxy(0,0);
    lcd_ubyte(ir_fl);
    lcd_setxy(1,0);
    lcd_ubyte(ir_hl);
}*/

    //Thunder faehrt immer
    while(ende == 0)
    {
        //Startschuss
        if (analog(8) < (wert_licht_l - 35)) //25
        {
            fahren = 1;
            //Seite ist auf "links" gesetzt
            links = 1;
        }

        //Startschuss
        if (analog(14) < (wert_licht_r - 35)) //25
        {
            fahren = 1;
```

```
            //Seite ist auf "rechts" gesetzt
            links = 0;
        }

        //Servomotor fuer Lenkung zuruecksetzen
        servo_arc(0, 77);

        //Servomotor fuer Akteur zuruecksetzen - normal:130
        servo_arc(2, 90);

        while(fahren == 1)
        {
            //Richtung einstellen
            motor_richtung(0, 0);
            //losfahren
            motor_pwm(0, 10);

            //Start ist Links!
            if (links == 1)
            {
                //geradeaus fahren und 3 Linien zaehlen
                linien_ueberfahren_r(3, 50);

                //nach rechts drehen und diagonal nach hinten zu den Atomen fahren
                servo_arc(0, 59); //rechts
                //Abbruchbedingung fuer das Lenken (rechter Optokoppler findet Linie)
                linien_ueberfahren_r(1,50);
                servo_arc(0, 77); //gerade

                sleep(1000);
                //Wand von der Diagonale kommend Modus 0
                wand_suchen(0,links);

                //fahren
                motor_pwm(0, 8);

                //nach links, um parallel zu den Atome zu fahren und Akteur ausfahren
                //Wand parallel zu den Atomen suchen Modus 1
                wand_suchen(1,links);

                //an der Wand ausrichten TODO
                ausrichtenWandRechts();

                //Variablen zum Beenden des Programms setzen
                fahren = 0;
                ende = 1;

            }
            else if (links == 0)//TODO rechts
            {

                //geradeaus fahren und 3 Linien zaehlen
                linien_ueberfahren_r(3, 50);

                //nach links drehen und diagonal nach hinten zu den Atomen fahren
                servo_arc(0, 96); //rechts
                //Abbruchbedingung fuer das Lenken (linker Optokoppler findet Linie)
                linien_ueberfahren_l(1,50);
                servo_arc(0, 77); //gerade

                sleep(1000);
                //Wand von der Diagonale kommend Modus 0, Richtung = rechts
                wand_suchen(0,links);

                //fahren
                motor_pwm(0, 8);

                //nach links, um parallel zu den Atome zu fahren und Akteur ausfahren
                //Wand parallel zu den Atomen suchen Modus 1
                wand_suchen(1,links);

                //an der Wand ausrichten TODO
                ausrichtenWandLinks();
```

```
                //Variablen zum Beenden des Programms setzen
                fahren = 0;
                ende = 1;

            }
        }
    }

    //Servomotor fuer Akteur zuruecksetzen
    servo_arc(2, 90);
    //Motor stoppen
    motor_pwm(0, 0);
    //Servomotor fuer Lenkung zuruecksetzen
    servo_arc(0, 77);

    // Die folgende Endlosschleife "erzeugt" das Programmende
    while(1);
}
```