UNIVERSITÄT SIEGEN

Theorie
und Praxis
für die Gesellschaft
von morgen

Fachbereich 12 – Elektrotechnik und Informatik
Fachgruppe Echtzeit Lernsysteme
Prof. Dr.-Ing. K.-D. Kuhnert

C-Programmierpraktikum für Elektrotechniker, Versuch „Pawlowscher Hund“

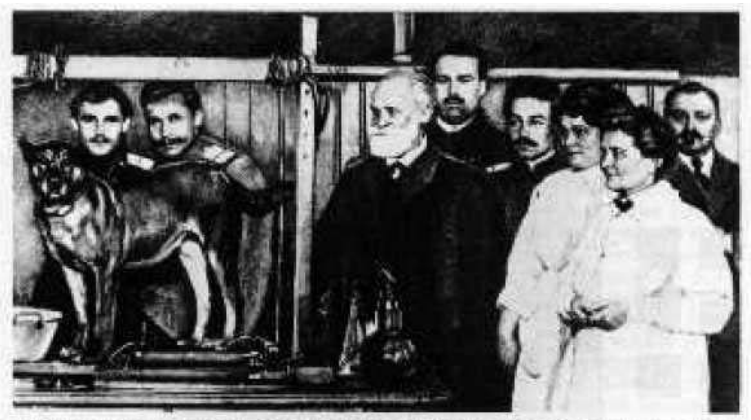
Zusammenfassung

Programmieren Sie einen mobilen Roboter des Typs Lego Mindstorm, so daß dieser einen bedingten Reflex erlernen kann.

Einleitung

Bedingte Reflexe kommen sehr häufig in der Natur vor. Als erster Mensch hat diesen Reflex Iwan Petrowitsch Pawlow nachgewiesen.

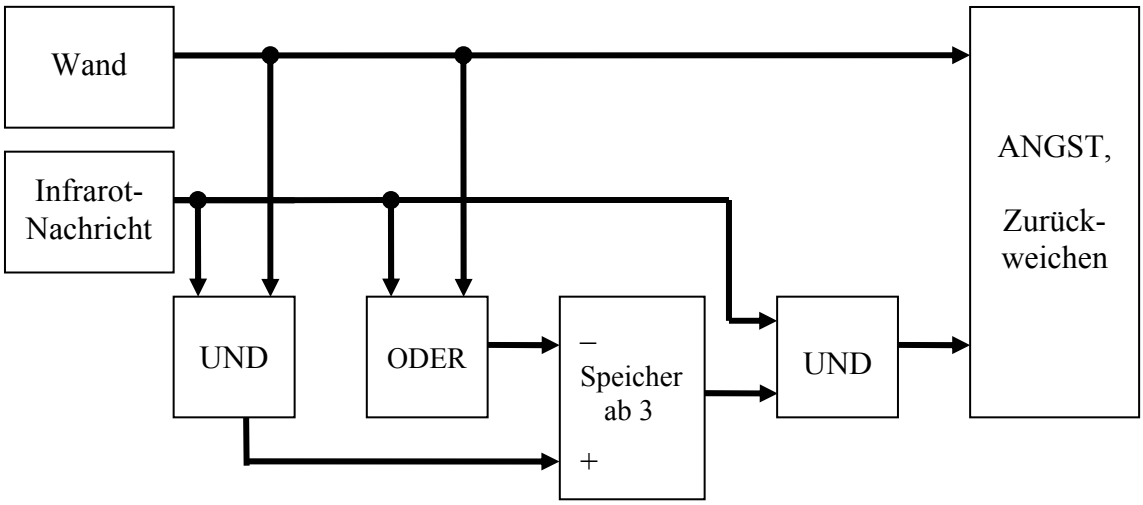
Iwan Petrovitsch Pawlow (1849-1936) wurde berühmt durch die Erforschung des konditionierten oder bedingten Reflexes bei Hunden. Wenn ein Hund Futter riecht, wird sein Speichelfluss ausgelöst. Diese Reaktion nennt man einen unbedingten Reflex. Unbedingte Reflexe sind angeborenen oder auch instinktive Reflexe. Durch Versuche an Hunden bewies Pawlow, dass man den unbedingten Reflex des Futterriechens verbunden mit dem Speichelfluss auch auf einen anderen Reitz umlernen kann. Immer dann, wenn der Hund zu fressen bekam, erklang eine Glocke. Nach einiger Zeit hatte der Hund den Klang der Glocke so sehr mit der Futtergabe in Verbindung gebracht, dass allein das Auslösen dieses Geräusches ausreichte, den Speichelfluss des Hundes auszulösen. Diesen Prozess des Trainings nannte Pawlow „Konditionierung“. Dieser neu erlernte Reflex wird auch als bedingter Reflex bezeichnet. Auf dem folgenden Foto ist Pawlow mit einem seiner Hunde zu sehen.



Iwan Pawlow während eines Versuchs mit einem Hund

Aufgabenstellung

Für diesen Versuch wird angenommen, daß der Roboter „Angst vor einer Wand“ hat und vor solchen Wänden zurückweicht. Das Zurückweichen ist der unbedingte Reflex. Er wird durch das Erkennen einer Wand mit Hilfe eines Tastsensors ausgelöst. Das Ertönen der Glocke aus der vorangehenden Beschreibung wird durch das Senden einer Infrarotnachricht an den Roboter nachgeahmt. Wird nun die Infrarotnachricht genau in dem Moment ausgelöst, in dem der Roboter eine Wand erkennt, soll der Roboter sich das Zusammentreffen dieser Ereignisse merken. Treten beide Ereignisse mehrere Male hintereinander zusammen auf, soll der Roboter das Empfangen einer Infrarotnachricht mit dem Erkennen einer Wand gleichsetzen und zurückweichen – sogar, wenn bei neuerlichem Senden des Signals noch gar keine Wand erkannt wurde. Empfängt der Roboter allerdings mehrfach eine Infrarotnachricht, ohne daß eine Wand erkannt wurde, soll die Verbindung dieser beiden Reize wieder aufgehoben werden. Eine einfache Möglichkeit dies zu realisieren, ist in dem nachfolgendem Diagramm verdeutlicht.

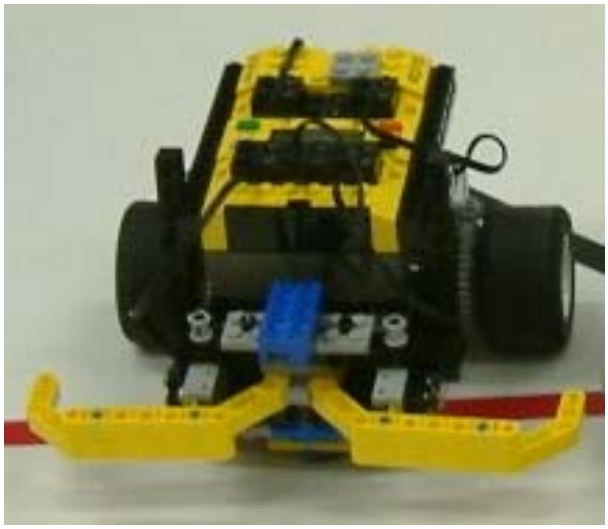


Wird eine Wand erkannt, erfolgt immer der unbedingte Reflex „Zurückweichen“. Die „UND“ Verknüpfung steht für gleichzeitiges Erkennen einer Wand und dem Empfang der IR-Nachricht. Trifft diese Verknüpfung zu, wird dem Speicher eine „1“ addiert. Zu Beginn des Versuchs ist der Wert des Speichers „0“. Dieser Wert soll auch nie unterschritten werden. Wird eine IR-Nachricht ohne Erkennen einer Wand gesendet, subtrahiert der Speicher eine „1“. Hat der Speicher den Wert „0“, bleibt dieser Wert bestehen. Sobald der Speicher den Wert „3“ erreicht hat, soll der Roboter den bedingten Reflex erlernt haben. Je höher dann der Wert des Speichers ist, das heißt, je öfter beide Reize gleichzeitig auftreten, desto langsamer kann er diesen bedingten Reflex auch wieder verlernen. Durch die „UND“ Verknüpfung zwischen dem Wert des Speichers und dem Empfang der IR-Nachricht wird gewährleistet, dass der bedingt Reflex nur ausgelöst wird, sobald der Roboter diesen erlernt hat.

Es wird das am Ende dieser Beschreibung abgedruckte Programm zur Verfügung gestellt, das bereits Funktionen zur Robotersteuerung, Sensorabfrage und Erkennung von Wänden enthält. Dieses soll um die oben beschriebene Funktionalität erweitert werden.

Der Roboter und die Entwicklungsumgebung

Der Roboter sieht etwa so aus:



Aufgebauter Roboter mit Stoßdämpfer

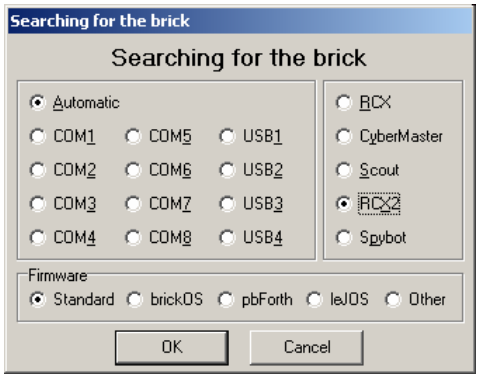
Batterieblock mit Mikroprozessor und anschließbare Sensoren und Aktoren

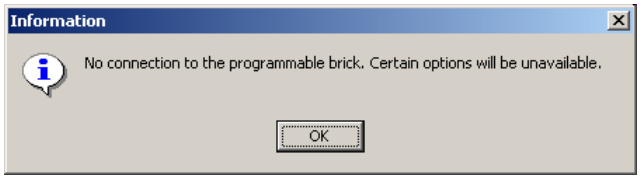
Er besitzt eine 8-bit-CPU mit 16 MHz von Hitachi, 16 KB internen ROM-Speicher, 32 KB statischen RAM-Speicher davon 6 KB für eigene Programme, serielle Ein-/Ausgabe, 10 Bit Analog- zu Digitalkonvertierung und integrierte Timer. Über drei Ausgänge kann der RCX Motoren oder Leuchten ansteuern. Er besitzt ebenfalls drei Eingänge, an die unterschiedliche Sensoren angeschlossen werden können.

Programmiert wird der Roboter in der Sprache *Not Quite C*, kurz NQC. Diese enthält die einfachsten C-Ausdrücke zur Programmierung von Schleifen und Funktionen. Außerdem gibt es noch ein paar zusätzliche Befehle, mit denen die Sensoren abgefragt, die Motoren bewegt und parallel ablaufende Prozeduren gestartet werden können. Im Gegensatz zu Standard-C sind in NQC alle Variablen vom Typ Integer (16 Bit). Pointer gibt es nicht. Die Zahl der Variablen ist auf etwa 30 begrenzt. Eine vollständige Dokumentation bietet das *NQC Programmer's Guide* von Dave Baum. Es kann unter http://bricxcc.sourceforge.net/nqc/doc/NQC_Guide.pdf heruntergeladen werden. Anstelle von langatmigen Erklärungen zur NQC-Syntax, sei hier auf das Beispielprogramm am Ende dieser Beschreibung abgedruckt.

Als Editor für NQC-Programme dient das *Bricx Command Center*, abgekürzt *Bricxcc*. Dieses ermöglicht neben der Eingabe des Programms auch dessen Übersetzung in eine für den Roboter verständliche Form und den Upload des Programms auf den Roboter. Dies geschieht über einen Infrarotsender („Infrarot-Tower"), der an der seriellen Schnittstelle des Rechners angeschlossen ist.

Vor dem Start des Bricxcc sollte der Roboter eingeschaltet vor dem Infrarot-Tower stehen, da ansonsten viele Funktionen von Bricxcc nicht zur Verfügung stehen. Nach dem Start des Programms erscheint das Fenster „Searching for the brick" (s. Abb., links).

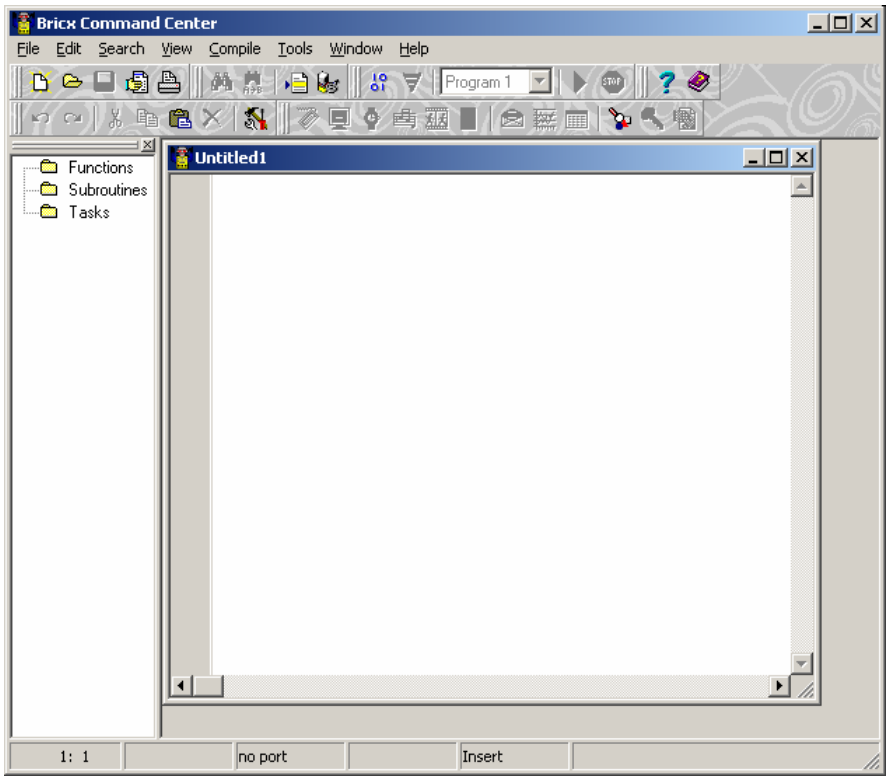




Nachrichtenfenster des Bricx Command Center

Ist der Anschluß des Infrarot-Towers bekannt, sollte er direkt ausgewählt werden, da es sein kann, dass der Roboter sonst nicht gefunden wird. Das Kürzel „RCX“ auf der rechten Seite bedeutet übrigens „Robotic Command Explorer“ und bezeichnet hier den Roboter. Falls ein RCX 1.0 verwendet wird, muss man trotzdem die Option RCX2 ankreuzen, da hier die Firmware des RCX 2.0 verwendet wird. Diese Firmware läuft auch auf dem RCX 1.0 ohne Probleme. Als Firmware wird Standard angekreuzt. Hat man alles berücksichtigt, öffnet sich dann das Programm Bricxcc.

Sollte das Fenster „Information“ (s. Abb., rechts) erscheinen, muss man überprüfen, ob der RCX eingeschaltet ist und nah genug an dem angeschlossenem Infrarot-Tower steht. Dieser muss ebenfalls an der richtigen Schnittstelle angeschlossen sein. Ist alles in Ordnung, sollte sich die Entwicklungsumgebung öffnen (nächste Abbildung).



Die Entwicklungsumgebung Bricxcc

Im Hintergrund zu dieser Entwicklungsumgebung öffnet sich noch ein weiteres Fenster. Dieses Fenster zeigt die am häufigsten verwendeten Befehle in NQC an. Es erleichtert später das Schreiben des Programmcodes, da die Befehle durch Anklicken automatisch in den Programmcode eingefügt werden.

Als nächstes sollte nun die Firmware auf den RCX übertragen werden. Dazu wird der Menüpunkt *Tools* → *Download Firmware* ausgewählt und aus dem Verzeichnis *Firmware* die Datei *firm0328.lgo* angeklickt. Der Download kann einige Minuten in Anspruch nehmen. Nach dem Download ist der RCX bereit, programmiert zu werden. Es ist eine gute Idee, den Roboter und den Infrarot-Tower während der Datenübertragung mit einem Pappkarton abzudecken. Dadurch wird die ungewollte Datenübertragung an andere Roboter im gleichen Raum unterbunden.

Mit dem Menüpunkt *Compile* → *Compile* wird ein Programm auf Fehler überprüft und für die Übertragung auf den Roboter vorbereitet. Mit dem Menüpunkt *Compile* → *Download* wird die Übertragung gestartet.

Das Senden von Infrarotnachrichten

Infrarotnachrichten können Zahlen von 0 bis 255 sein. Sie werden in NQC durch den Befehl *Message()* abgefragt. Da der Befehl den Wert Null zurückgibt, falls gerade keine Nachricht empfangen wurde, wird von dem Senden der Nachricht „0“ abgeraten. Eine Nachricht wird bis zum Aufruf des Befehls *ClearMessage()* gespeichert. Zum Senden einer Nachricht wählen Sie im Bricxcc den Menüpunkt *Tools* → *Send Messages*. Es erscheint das folgende Fenster.

Fenster zum Senden von Infrarotnachrichten

Nun können die zu sendenden Zahlen direkt angeklickt werden oder in dem Eingabefeld unten links eingetragen und mit dem „Send“-Button verschickt werden. Es ist

Beispielprogramm in NQC

Das folgende Programm zeigt eine Robotersteuerung in NQC. Der Roboter fährt nach dem Programmstart zunächst geradeaus. Erkennt der Roboter mit Hilfe seines Tastsensors eine Wand, fährt er ein Stück rückwärts und wendet dann. Daraufhin fährt er wieder geradeaus. Außerdem reagiert er auf eine bestimmte Infrarotnachricht. Wird diese erkannt, gibt er die Zeitdauer seit dem letzten Zusammenstoß mit einer Wand aus. Diese wird mit Hilfe von Timer 0 gemessen. Bei einem Zusammenstoß mit einer Wand wird dagegen die Zeit seit der letzten Infrarotnachricht gemessen. Dies geschieht mit Hilfe von Timer 1.

Die Ansteuerung des Motors geschieht über die parallel ablaufenden Tasks *Move()* und *Behaviour()*. Der Task Behaviour ließt ständig die Variable *_Behaviour* aus. Diese gibt an, ob der Roboter geradeausfahren oder zurückweichen soll. Das entsprechende Verhalten wird daraufhin in einzelne, einfache Bestandteile zerlegt, z.B. für das Wendemanöver in die Bestandteile „Rückwärts fahren“, „Wenden“, und „Weiterfahren“. Der gerade auszuführende Bestandteil wird in die Variable *_Movement* geschrieben. Der Task *Move()* ließt diese Variable ständig aus und gibt passende Befehle an die Motoren.

```
// Martin.Stommel@gmx.de, 5.4.2006

// Zusammengesetzte Bewegung des Roboters
int _Behaviour;       // 0 -> Fahren, 1 -> Zurückweichen

// Elementare Bewegung des Roboters
int _Movement;        // 0 -> Halt, 1 -> Geradeaus, 2 -> Rückwärts, 3 -> Drehen


// Einfache Roboterbewegungen ausführen: Geradeausfahren, drehen, etc.
task Move()
{
    // Ausgang A ist am linken Motor angeschlossen,
    // Ausgang C am rechten.

    // Erstmal sicher stehen.
    SetPower( OUT_A + OUT_C, 1 );           // Niedrige Motorkraft
    SetDirection( OUT_A + OUT_C, OUT_FWD ); // Beide Motoren in Vorwärtsgang
    SetOutput( OUT_A, OUT_OFF );            // Beide Motoren noch ausgeschaltet
    SetOutput( OUT_C, OUT_OFF );
    int _PreviousMovement = 0;

    // Immer wieder...
    while( 1 )
    {
        // Änderungen der beabsichtigten Bewegung erkennen
        if( _Movement != _PreviousMovement)
        {
            _PreviousMovement = _Movement;

            // Neue Bewegung einstellen
            if( 0 == _Movement )
            {
                // Anhalten
                SetPower( OUT_A + OUT_C, 1 );
                SetDirection( OUT_A + OUT_C, OUT_FWD );
                SetOutput( OUT_A, OUT_OFF );
                SetOutput( OUT_C, OUT_OFF );
            }
            else if( 1 == _Movement )
            {
                // Geradeaus
                SetPower( OUT_A + OUT_C, 1 );
                SetDirection( OUT_A + OUT_C, OUT_FWD );
                SetOutput( OUT_A, OUT_ON );
                SetOutput( OUT_C, OUT_ON );
            }
            else if( 2 == _Movement )
            {
                // Rückwärts
                SetPower( OUT_A + OUT_C, 1 );
                SetDirection( OUT_A + OUT_C, OUT_REV );
                SetOutput( OUT_A, OUT_ON );
                SetOutput( OUT_C, OUT_ON );
            }
            else if( 3 == _Movement )
            {
                // Rückwärts
                SetPower( OUT_A + OUT_C, 1 );
                SetDirection( OUT_A, OUT_REV );
                SetDirection( OUT_A, OUT_FWD );
                SetOutput( OUT_A, OUT_ON );
                SetOutput( OUT_C, OUT_ON );
            }
        }
    } // while 1
} // task Move
```

```
// Zusammengesetzte Roboterbewegung ausführen
task Behaviour()
{
    while( 1 )
    {
        if( _Behaviour )
        {
            // Zurückweichen

            // erst ein Stück zurück
            _Movement = 2;
            Wait( 60 );    // 60 * 10ms warten

            // dann eine Weile drehen
            _Movement = 3;
            Wait( 110 );    // 110 * 10ms warten

            // dann wieder normal weiterfahren
            _Behaviour = 0;
        }
        else
        {
            // Fahren
            _Movement = 1;
        }
    } // while 1
} // task Behaviour


// Tastsensor auswerten (Zusammenstöße mit einer Wand)
task EvaluateBumper()
{
    // Sensoreingang 3 als Tastsensor konfigurieren
    SetSensor( SENSOR_3, SENSOR_TOUCH );

    while( 1 )
    {
        // Tastsensor auslesen
        if( SENSOR_3 )
        {
            // Taster wurde gedrückt => zurückweichen
            _Behaviour = 1;

            // Timer 0 für die Zeit seit dem letzten Zusammenstoß zurücksetzen
            ClearTimer( 0 );

            // Die Zeit seit der letzten Infrarotnachricht
            // auf dem LCD-Display des Roboters anzeigen
            int Time = Timer( 1 );
            SetUserDisplay( Time, 0 );
        }
    } // while 1
} // EvaluateBumper


// Infrarotnachricht auswerten
task EvaluateInfrared()
{
    while( 1 )
    {
        // Infrarotnachricht auslesen
        int InfraredMessage = Message();

        // Dieser Roboter reagiert nur auf die Nachricht '4'
        if( 4 == InfraredMessage )
        {
            // Es wurde eine passende Infrarotnachricht empfangen

            // Timer 1 für die Zeit seit der letzten IR-Nachricht zurücksetzen
            ClearTimer( 1 );
```

```
            // Die Zeit seit der letzten Zusammenstoß auf
            // auf dem LCD-Display des Roboters anzeigen
            int Time = Timer( 0 );
            SetUserDisplay( Time, 0 );

            // Empfangene Nachricht löschen => Platz für neue
            ClearMessage();
        }
    }
} // EvaluateInfrared


task main()
{
    // definierten Anfangszustand schaffen
    _Behaviour = 0;  // Verhalten 'Fahren'
    _Movement = 0;   // Halten

    // Es gibt zwei Timer, die in Schritten von 10ms hochgezählt werden.
    // Timer 0 soll die Zeit seit dem letzten Zusammenstoß anzeigen.
    // Timer 1 soll die Zeit seit der letzten Infrarotnachricht anzeigen.
    SetTimer( 0, 1000 );   // Timer auf 1000 setzen, d.h. der letzte Zusammen-
    SetTimer( 1, 1000 );   // stoß oder die letzte Nachricht ist lange her.

    // Diverse parallel ablaufende Tasks starten.
    start Move;
    start Behaviour;
    start EvaluateBumper;
    start EvaluateInfrared;

    // Nie enden.
    while( 1 )
    {
    }
} // task main
```