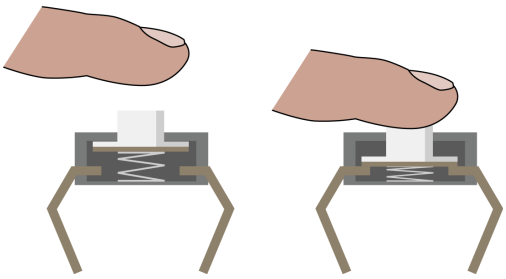
OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

NNXT - Tutorials

Tutorial #03 - Der Tastsensor und Funktionen zur Zeitmessung

Der Tastsensor

Der Tastsensor dient dazu, eine Berührung des Roboters mit einem Gegenstand oder einem Benutzer zu detektieren. Ein Taster hat einen von zwei Zuständen: entweder er ist heruntergedrückt oder er ist nicht heruntergedrückt. Technisch wird dies durch die Herstellung bzw. Unterbrechung eines elektrischen Stromflusses aufgebaut. Der nicht berührte Zustand wird dabei durch eine Feder hergestellt, die beim reduzieren der Kraft auf den Knopf mittels der Federkraft als Gegenkraft den Knopf wieder in die nicht berührende Position bringt. Die folgende Abbildung stellt das Funktionsprinzip eines Tasters dar (beachten Sie den Unterschied zu einem Schalter, der den durch Krafteinwirkung hergestellten Zustand beibehält):



Das Grundprinzip ist dabei, dass der heruntergedrückte Knopf eine elektrische Leitfähigkeit zwischen den beiden seitlich ausgehenden Leitern herstellt. Diese Leitfähigkeit wird durch eine elektronische Schaltung überprüft und kann von der Recheneinheit abgelesen werden. Wird der Knopf nicht mehr heruntergedrückt, so wird die leitende Verbindung der beiden seitlichen ausgehenden Leiter unterbrochen, da die Feder den Knopf inklusive Verbindungsleiter wieder nach oben drückt.

Der Lego-Roboter-Bausatz verfügt über derartige Taster, die folgendermaßen aussehen:

Wie der Taster tatsächlich elektrisch aufgebaut ist, werden wir uns in einem späteren Teil genauer anschauen. Hier genügt zunächst die Information, dass der Taster gedrückt oder losgelassen ist. Der Taster muss an einen der 4 Sensor-Ports Port 0 bis Port 3 angeschlossen werden und kann mittels der API der NNXT-Library programmiert werden.

Die Taster-API

Bevor wir den Tastsensor benutzen können, muss zunächst der Port, an dem der Tastsensor angeschlossen ist, konfiguriert werden. Dies erfolgt einmalig am Beginn des Programmes durch folgenden Funktionsaufruf, wobei ***Port_X*** durch **Port_0**, **Port_1**, **Port_2** oder **Port_3** zu ersetzen ist, je nachdem an welchem Port Sie den Sensor angeschlossen haben:

```
    SensorConfig(Port_X, SensorTouch);
```

Anschließend kann der Sensor verwendet werden. Um Entfernungswerte vom Sensor zu bekommen, können Sie die folgende Funktion benutzen:

```
    sensor_error_t Touch_Clicked(sensorport_t port,
sensor_touch_clicked_t *clicked);
```

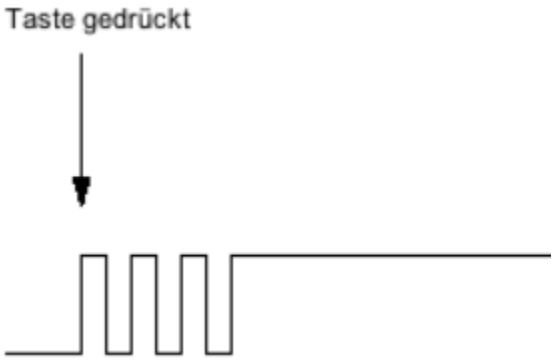
Wie beim Ultraschallsensor bekommen Sie den Wert des Sensors über einen Call-by-Reference Parameter (hier: **clicked**) und die Funktion liefert einen Fehlercode. Der Wert des Sensors hat hier eigenen eigenen Typ, nämlich **sensor_touch_clicked_t**, welches ein Aufzählungstyp mit den möglichen Werten **SensorTouch_clicked** und **SensorTouch_released** ist. Weitere Werte existieren für diesen Aufzählungstyp nicht und die Bedeutung ergibt sich aus dem Namen der Werte.
Ein anderer Fehlercode als **sensor_error_NoError** wird bei Tastsensoren nicht auftreten, da es a) keine zeitliche Limitierung zwischen zwei Abfragen des Sensors gibt und b) die Ports standardmäßig als **SensorTouch** ausgelegt sind. Demzufolge können Sie den Rückgabewert dieser Funktion ignorieren (aus Gründen einer Standardisierung liefert diese Funktion trotzdem einen Fehlercode zurück). Zur Verdeutlichung folgender Beispiel-Code eines Programmes, das den Tastsensor (angeschlossen an Port 0) ausliest und dessen Zustand auf dem Display ausgibt:

```
int main() {
    sensor_touch_clicked_t touch;
    SensorConfig(Port_0, SensorTouch);
    while(1) {
       Touch_Clicked(Port_0, &touch);
```

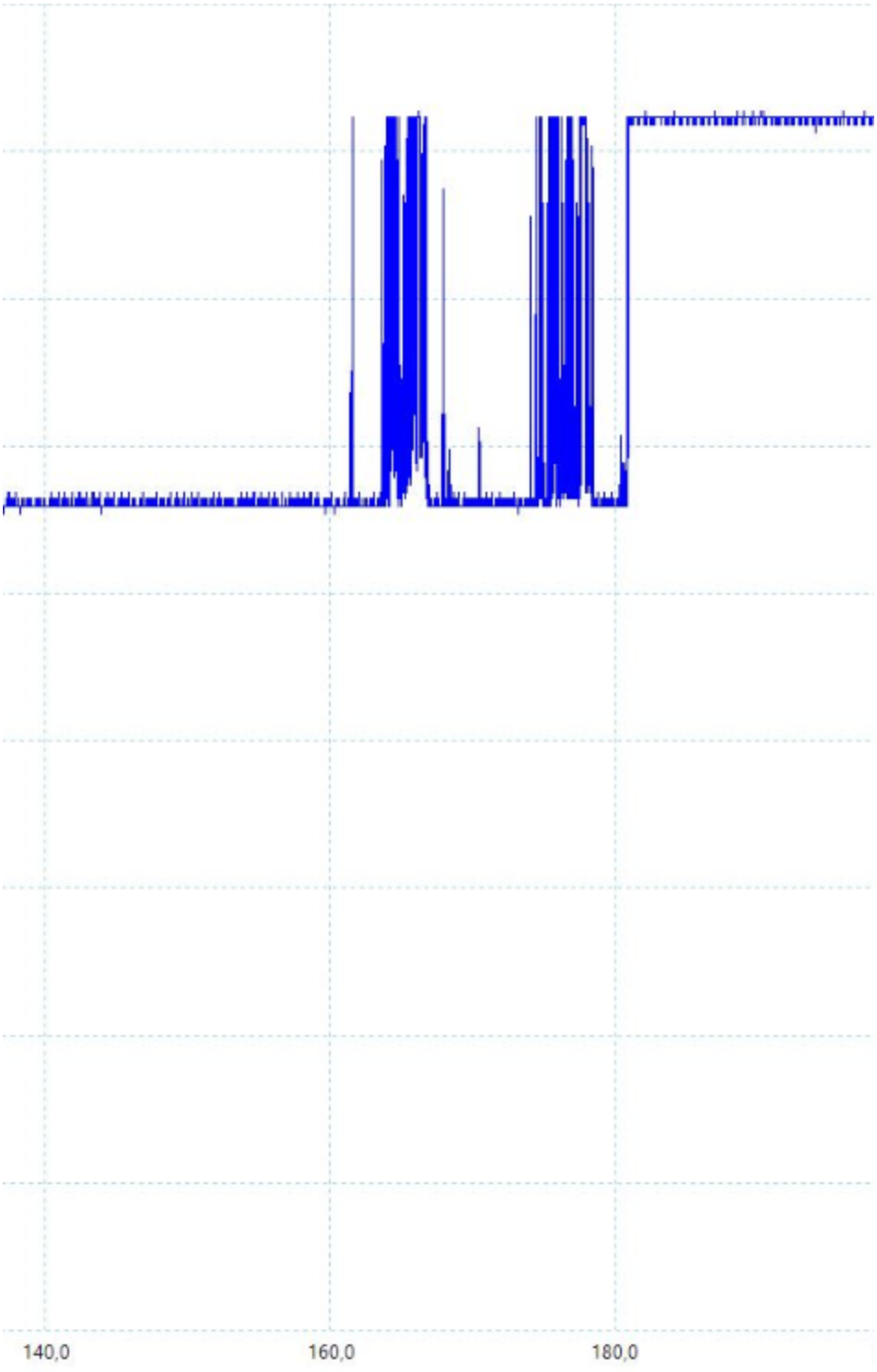
```
        if (touch == SensorTouch_clicked)
           NNXT_LCD_DisplayStringAtLine(0, "Taster gedrueckt");
         else
           NNXT_LCD_DisplayStringAtLine(0, "Taster losgelassen");
    }
}
```

Prellen des Tasters

Prinzipiell funktioniert dieses Programm, allerdings treten bei Tastsensoren und Schaltern durch physikalische Effekte Störungen auf. Dies liegt daran, dass beim Herunterdrücken der Leiter des Knopfes auf die Kontakte der seitlich ausgehenden Leiter (siehe Abbildung oben) prallt. Physikalisch findet hier ein elastischer Stoß zwischen 2 Objekten statt, wie Sie es aus dem Physikunterricht kennen und wie man es mit einem fallenden Basketball sehr schön verdeutlichen kann. Beim Aufprall wird daher wegen der Impulsübertragung und Vibrationen der Anschlusskontakte durch den Stoß der Knopf ganz leicht und sehr schnell auf und ab prellen, wie ein Basketball, der durch die Gravitationskraft nach unten gezogen wird, beim Aufprall auf den Boden auf und ab prellt. Dadurch ergibt sich ein sehr schneller Wechsel zwischen geöffneten und geschlossenen Kontakten, bis der Kontakt letztendlich aufliegt und den Dauerhaften Zustand erreicht. Die folgende Grafik veranschaulicht diesen Effekt:



Diese Gedrückt/Nichtgedrückt Variation geschieht normalerweise recht hochfrequent. Da wir aber im obigen Beispielprogramm den Tastsensor maximal schnell abfragen, werden wir diesen Effekt beobachten können. Konkrete Messungen für den Lego-Taster ergeben dabei das folgende Verhalten:



In der Abbildung sehen Sie den zeitlichen Verlauf des Taster-Signals mit einer Zeitauflösung in Millisekunden. Man kann an der Zeitskala unten erkennen, dass der Taster etwa zur Zeit 160ms gedrückt wurde. Dann folgt das Prellen und nach etwa 180ms stellt sich der stabile Zustand des Tastersignals ein. Das bedeutet, wenn wir dieses Prellen nicht mit detektieren wollen, dass wir mindestens 20ms zwischen 2 Messungen warten müssen. Demzufolge müssten wir den Programmcode von oben abändern und am Ende der **`while`**-Schleife ein **`Delay`** von 20ms einfügen müssten (vgl. Vorlesung). Damit sollte dann das Prellen nicht erkennbar sein (gleichzeitig begrenzt das Prellen und die notwendige Verzögerung natürlich auch die maximal zu detektierende Frequenz des Taster-Signals, vgl. Abtast-Theorem aus der Vorlesung).

Zeitmessung

Bei vielen Aufgaben, die ein Roboter durchführen muss, ist es wichtig die Zeit zu messen, beispielsweise wenn der zeitliche Abstand von zwei Drücken auf einen Taster oder die Länge eines Tasterdrucks gemessen werden soll. Wie in den vorangegangenen Programmen gesehen, können Wartezeiten über die **Delay**-Funktion erzeugt werden. Man könnte also zur Messung einer Zeit einen Zähler verwenden, der die Anzahl an **Delay**-Aufrufen zwischen zwei beobachtbaren Ereignissen zählt. Angenommen, wir wollen die Länge eines Tastendruckes feststellen: dann starten wir einen Zähler, sobald das Programm erstmalig den Druck erkennt und zählen so lange hoch, bis der Tastendruck nicht mehr erkennbar ist. Mit der bekannten Größe von **Delay** kann dann die Zeitspanne ausgerechnet werden. Im Beispielcode, bei dem während des Tastendrucks die bisher verstrichene Zeit in Millisekunden ausgegeben wird, würde das folgendermaßen aussehen:

```
int main() {
    sensor_touch_clicked_t touch;
    char msg[20];
    uint16_t zaehler = 0;
    SensorConfig(Port_0, SensorTouch);
    while(1) {
       Delay(20);
       Touch_Clicked(Port_0, &touch);
       if (touch == SensorTouch_clicked)
          zaehler++;   // alle 20ms um eins erhöhen
        else
          zaehler = 0;
        if (zaehler > 0) {
          sprintf(msg,"Dauer bisher:%d",(int)zaehler*20);
          NNXT_LCD_DisplayStringAtLine(0,msg);
        } else
          NNXT_LCD_DisplayStringAtLine(0,"                ");
    }
}
```

Ist die auf diese Weise gemessene Zeit im Bereich von zig Millisekunden, so ist die Methode aus dem obigen Programm erlaubt. Wenn die Zeitspanne allerdings im unteren einstelligen Millisekundenbereich oder sogar in den Mikrosekundenbereich geht, so gibt es hier ein Problem: Zwar wissen wir, dass die **Delay**-Funktion 20ms wartet, aber wir wissen nicht, wie lange die Ausführungszeit des übrigen Codes des Schleifenrumpfes der while-Schleife dauert. Insbesondere Display-Funktionen und sprintf sind komplexe Funktionen, die eine gewisse Laufzeit haben. Diese Zeit kommt zu den 20ms des Delay hinzu, so dass die Berechnung der Zeit mit jeder Schleifeniteration ungenauer wird. Sei die Zeit, die der Schleifenrumpf verbraucht X ms, dann ergibt die Messung (zaehler*20)ms, in Wirklichkeit sind aber (zaehler*(20+X))ms verstrichen. Bei komplexen Programmen mit vielen if-else-Verzeigungen kann zudem die Größe X nicht angeben werden, da diese - abhängig vom Kontext - in jeder Schleifeniteration anders sein kann (unterschiedliche Pfade im Programm). Folglich ist die oben vorgestellte Zählmethode ungenau und sollte nur bei sehr einfachen Programmen, bei denen X sehr klein verglichen mit der **Delay**-Zeit ist, verwendet werden. In allen anderen Fällen sollte man ein alternatives, genaueres Konzept verwenden.

Die NNXT-Library bietet ein solches Konzept an, in dem der NNXT eine interne Uhr enthält, die unabhängig von etwaigen Programmausführungen läuft. Diese Uhr wird in Form eines Zählers wie oben bereitgestellt, der jede Millisekunde um 1 erhöht wird. Da diese Erhöhung aber nicht innerhalb eines Programmes erfolgt, sondern im Prinzip von der Hardware des Systems durchgeführt wird, ergibt sich keine Ungenauigkeit, wie im oben diskutierten Fall. Den aktuellen Stand dieses Zählers können Sie mithilfe der folgenden Funktion auslesen:

```
uint32_t GetSysTime();
```

Der Zähler wird beim Systemstart des Roboters auf 0 gesetzt und beginnt dann zu zählen, jede Millisekunde um 1 höher. Einen Überlauf müssen Sie übrigens nicht befürchten, da der Zähler eine vorzeichenlose 32-Bit Variable ist, mit der von 0 bis maximal 4294967295 Millisekunden gezählt werden kann, bevor ein Überlauf auftritt. Dies entspricht etwa 1193 Stunden, was deutlich über der Lebenszeit der Batterien im Roboter und auch deutlich über der Länge einer Übungsstunde liegt.

Wie das obige Programm zur Messung der Länge eines Tastendrucks nun unter Verwendung der **`GetSysTime`**-Funktion aussieht, sei dem geneigten Leser zur einfachen Übung überlassen. Ein Hinweis hierzu: Da grundsätzlich nicht bekannt ist, wann genau das erstmalige Drücken des Tasters erfolgt und damit der Zählerstand der internen Uhr nicht vorab bestimmt werden kann, müssen Sie den von **`GetSysTime`** gelieferten Zählerstand beim erstmaligen Druck abspeichen und anschließend die Differenz von **`GetSysTime`** und dem abgespeicherten Zählerstand betrachten, um die Dauer ausrechnen zu können.