c't magazin für computer technik

Positionsbestimmung für den c't-Bot

Wo bin ich?

Praxis & Tipps | Praxis .. Uhr
Torsten Evers

Normalerweise liefert ein Maussensor nur Veränderungen in X- und Y-Richtung. Zusammen mit ein paar Annahmen über die Bewegungsfreiheitsgrade des c't-Bot und dessen Geometrie lassen sich daraus aber Ausrichtung und Position ableiten. Die Korrelation mit Werten anderer Sensoren steigert nicht nur die Genauigkeit, sondern liefert auch Informationen über die Umwelteinflüsse.

Bei allen bisherigen Aufgaben, die der c't-Bot zu lösen hatte, reagierte er nur auf seine Umwelt. Ihm fehlte eine zuverlässige Möglichkeit, Blickwinkel und Position zu ermitteln. Diese lassen sich jedoch mit ein wenig Mathematik und einigen Zeilen Code aus den Rohdaten der Rad-Encoder und insbesondere des optischen Maussensors ermitteln. Im Idealfall sollten Rad-Encoder und Maussensor äquivalente Werte liefern. Da sie aber ganz unterschiedliche Dinge beobachten - Drehung der Räder versus Bewegung über den Boden - kann man aus Abweichungen interessante Rückschlüsse über Umwelteinflüsse auf den c't-Bot ziehen. Blockiert beispielsweise ein Hindernis den Roboter, so drehen die Räder durch, der Maussensor liefert aber keine Positionsänderung mehr.

Der c't-Bot beobachtet nur Veränderungen und kann somit keine absolute Positionen ermitteln. Er bestimmt nur eine relative Position per Koppelnavigation. In kurzen Abständen bestimmt er die gefahrenen Strecken und Drehungen und verrechnet sie mit dem letzten Standort. Das ergibt eine neue Position und den aktuellen Blickwinkel. Allerdings addieren sich im Lauf der Zeit bei einem relativen Verfahren alle noch so kleinen Messfehler und die Abweichung von der absoluten Position nimmt zu.

Allgemein bezeichnet man die Bestimmung zurückgelegter Entfernungen durch Beobachtung der Radumdrehungen als Odometrie, abgeleitet vom griechischen hodós (Weg) und métron (Maß). Ein einfaches Beispiel für ein Odometer ist ein Tachometer, wie er im Auto Verwendung findet. Auch die Rad-Encoder des c't-Bots gehören zu dieser Gattung. Die kleinste mit ihnen messbare Entfernung liegt aufgrund des Raddurchmessers und der Rasterung der Encoder-Scheiben bei 3 mm.

Agiles Nagetier

Der optische Maussensor (Avago ADNS-2610) liefert deutlich genauere Werte. Er steckt auch in vielen PC-Mäusen und arbei-tet mit einer Auflösung von 0,635 mm (400 Counts Per Inch, CPI).

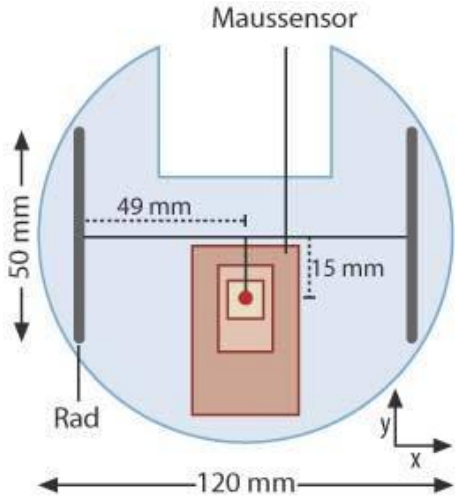
Eine integrierte 19x19-Pixel-CCD-Graustufenkamera beobachtet den Untergrund. Aus der Verschiebung nacheinander aufgenommener Bilder ermittelt der Sensor relative Bewegungen und gibt sie als Delta-X- (dX) und Delta-Y-Werte (dY) aus. Beim c't-Bot erlauben sie Rückschlüsse auf Bewegungen in Fahrtrichtung (Translation) und Drehungen (Rotation).

Die Register des Maussensors für dX und dY sind nur 8 Bit breit (-128 bis +127), das entspricht einer Wegstrecke von rund 8 mm. Fragt die Software den Sensor zu selten ab, kommt es zu Überläufen. Daher holt die Firmware sie häufig ab und kumuliert sie in den statischen Variablen `sensMouseX` und `sensMouseY`.

Die Linse des Sensors sitzt 1,5 cm hinter dem Mittelpunkt des Bots und in der Originalversion circa sechs mm über dem Boden. Dieser Abstand führt zu leicht unscharfen Bildern. Etwas Tuning und Tieferlegen wirkt Wunder. Informationen zur Feinabstimmung dazu kann die Software des c't-Bot aus den Registern des Sensors auslesen. Auch die von der Kamera aufgenommenen Bilder stehen zur Verfügung und können bei der Kalibrierung gute Dienste leisten. Dazu mehr im Kasten auf Seite 228.

Quo vadis?

Eine mechanische Eigenschaft oder genauer gesagt Beschränkung des Bots macht die Bestimmung der Drehung überhaupt erst möglich: Anders als eine PC-Maus kann der Bot aufgrund der Positionierung der Räder keine seitlichen Bewegungen ausführen. Richtungsänderungen geschehen immer in Form von Drehungen. Beobachtet der exzentrisch sitzende Maussensor eine seitliche Bewegung, muss eine Rotation sie verursacht haben. Unabhängig davon, wie oft man ihn abfragt, beobachtet der Sensor intern alle 661 µs die Bewegung und kumuliert die Einzelwerte. Selbst wenn der Bot nun mit maximaler Geschwindigkeit im Kreis dreht, sieht der Sensor bei jeder Messung nur winzige Ausschnitte aus dem Drehkreis. Für diese ist die Näherung mit einer Geraden akzeptabel. Obwohl also das Zentrum des Maussensors sich bei einem Vollkreis durchaus in X- und Y-Koordinaten der Welt verschiebt, misst der Sensor für seinen lokalen X-Wert näherungsweise nur die auf dem Kreisbogen abgefahrene Strecke. Die aktuelle Blickrichtung ergibt sich somit näherungsweise aus der Differenz zwischen dem aktuellen X-Wert des Maussensors und dem letzten gemessenen Wert:



Der Maussensor sitzt in der Mitte zwischen den Rädern, aber 15 mm hinter der Radachse des c't-Bot.

```
dX=sensMouseX - lastMouseX;
dHead=(float)dX*360.0/MOUSE_FULL_TURN;
heading_mou+=dHead;
lastHead+=dHead
```

Die errechnete Drehung addiert die Funktion `sensor_update()` aus der Datei sensor.c zum bisherigen Blickwinkel. Da dieser Wert später noch für die Berechnung der gefahrenen Geschwindigkeit Verwendung findet, summiert man die Winkeländerungen seit der letzten Auswertung zusätzlich in der Variable `lastHead` auf. Nach Korrektur von Überläufen liegt der Blickwinkel immer zwischen 0° und 360°:

```
while (heading_mou>=359)
heading_mou-= 360.0;
while (heading_mou<0)
heading_mou+= 360.0;
```

Diese und einige weitere Berechnungen erfolgen mit Hilfe von Fließkomma-Operationen. Das kostet zwar auf dem Mikrocontroller deutlich mehr Zeit als Rechnungen mit Integer-Werten, ist aber für die Genauigkeit unerlässlich. Denn bei Ganzzahlen ergäben sich mit jedem Rechenschritt erhebliche Rundungsfehler, die die Genauigkeit des Maussensors binnen weniger Messungen auffressen würden. Je öfter man die Position und Blickrichtung neu berechnet, desto stärker fällt dieser Effekt ins Gewicht. Da Funktionen und Verhalten wie `bot_turn()` oder `bot_drive_distance()` häufig präzise Informationen über die bereits zurückgelegte Strecke benötigen, um den Roboter an der gewünschten Stelle wieder anzuhalten, führt kein Weg an float-Variablen vorbei.

Eine Frage des Standpunktes

Aus dem aktualisierten Winkel und der gefahrenen Strecke ergibt sich die Positionsänderung. Die daraus resultierenden Koordinaten `x_mou` und `y_mou` beziehen sich auf die Position, an der der Bot beim Einschalten stand, und sind in mm angegeben. Der Maussensor arbeitet aber intern mit Zoll - der Faktor `25.4` gleicht das aus:

```
dY=sensMouseY-lastMouseY;
lastDistance+=dY*25.4/MOUSE_DPI;
dPos=(float)dY*cos(heading*PI/180)*25.4/MOUSE_CPI;
y_mou+=dPos;
dPos=(float)dY*sin(heading*PI/180)*25.4/MOUSE_CPI;
x_mou+=dPos;
```

Die gefahrene Strecke (`lastDistance`) bildet zusammen mit der Winkeländerung die Grundlage für die Berechnung der Geschwindigkeit. Um im nächsten Durchlauf wieder mit den Differenzen zu den aktuellen Sensorwerten arbeiten zu können, aktualisiert man zuletzt noch die Variablen `lastMouseX` und `lastMouseY`:

```
lastMouseX=sensMouseX;
lastMouseY=sensMouseY;
```

Alle 500 ms bestimmt `sensor_update()` aus den Änderungen der Encoder-Werte der einzelnen

Räder sowie den soeben besprochenen Werten des Maussensors die Geschwindigkeit des Bots. Die Encoder-Werte liefern:

```
v_enc_left= (((sensEncL – lastEncL) ∗ WHEEL_PERIMETER) / ENCODER_MARKS)/0.5;
v_enc_right= (((sensEncR – lastEncR) ∗ WHEEL_PERIMETER) / ENCODER_MARKS)/0.5;
```

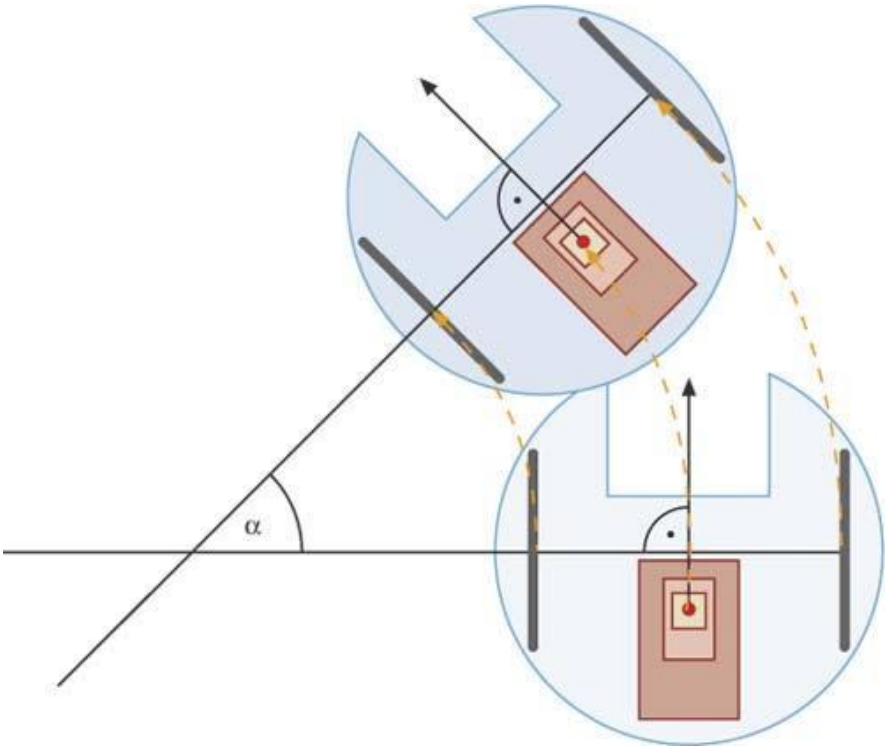
Da die Berechnung zwei Mal pro Sekunde erfolgt, normiert die Division durch 0.5 die Geschwindigkeit auf mm/s. Der Maussensor misst die Geschwindigkeit im Zentrum des Bots. Die gefahrene Strecke liegt bereits vor:

```
v_mou=lastDistance/0.5;
```

Leider sind die so gemessenen Geschwindigkeiten nicht direkt vergleichbar, da die der Rad-Encoder sich auf die beiden Räder beziehen. Die Umrechnung von `v_left` und `v_right`, auf einen zentralen Bezugspunkt stellt kein Problem dar:

```
v_enc=(v_enc_left+v_enc_right)/0.5;
```

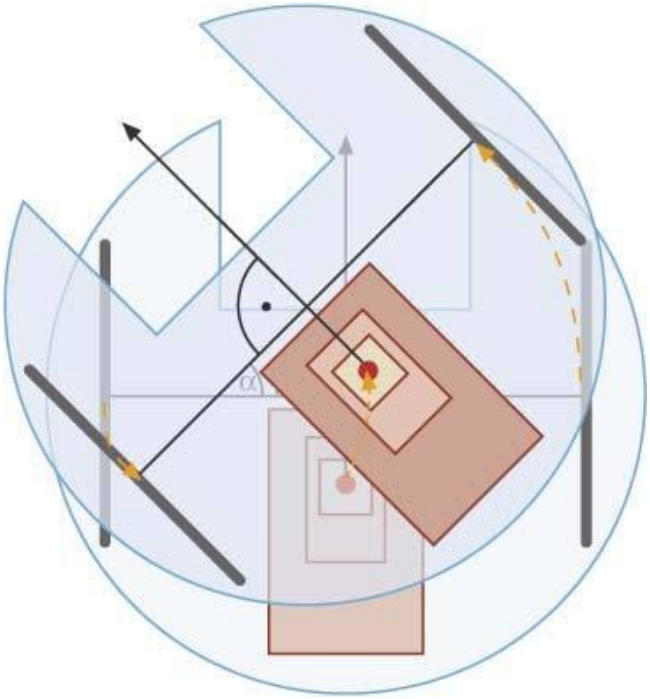
Das Aufteilen der zentralen Geschwindigkeit auf die beiden Räder birgt nicht nur mathematische Probleme, sondern kostet auch Rechenzeit. Möchte man jedoch beispielsweise die Motorregelung mit den Daten des Maussensors präzisieren, führt daran kein Weg vorbei.



Der c't-Bot dreht immer um einen Punkt, der auf der verlängerten Radachse liegt. Bei jeder Drehung überstreicht der exzentrische Maussensor einen Kreisbogen.

Bei jeder gleichförmigen Drehung des Bots liegt der Drehpunkt auf der verlängerten Radachse. Das kurze Messintervall erlaubt diese Näherung. Fahren beide Räder in dieselbe Richtung, dreht der Bot um einen Punkt außerhalb des Bots, bei gegenläufigen Rädern um einen Punkt zwischen den beiden Rädern. Die Räder und der Maussensor bewegen sich dabei auf konzentrischen Kreisbahnen um eben dieses Zentrum. Bei geradliniger Fahrt werden diese Kreisbahnen zu Geraden und die Geschwindigkeit der Räder stimmt mit der im Zentrum des Bots und damit auch

mit v_mou überein:



Drehen die Räder gegenläufig, liegt der Drehpunkt zwischen den Rädern. Je näher er dem Zentrum kommt, desto kleiner wird die Delta-Y-Komponente, die der Maussensor liefert.

```
if (lastHead==0) {
v_mou_left= v_mou;
v_mou_right= v_mou;
}
```

Bei Kurvenfahrten oder Drehungen ergibt sich die Geschwindigkeit eines Rades aus der auf dem jeweiligen Kreisbogen zurückgelegten Strecke und der Zeit zwischen zwei Messungen. Das Zentrum des Drehkreises ergibt sich als Schnittpunkt der zu Geraden verlängerten Radachsen an alter und neuer Position. Beide Geraden sind eindeutig definiert, da mit der Position des Roboters ein Stützpunkt und mit der Blickrichtung eine Normale bekannt sind. Die absolute Position dieses Punktes im Raum interessiert jedoch nicht, sondern nur die Distanz zum Zentrum des Bots.

Aus dem Tangens des Winkels beider Geraden ergibt sich deren Steigung. Für den Fall, dass einer von ihnen 90 oder 270 Grad beträgt, ist die Funktion nicht definiert. Da für die weitere Berechnung nur der Abstand des Drehpunktes vom Mittelpunkt des Bots wichtig ist, rechnet man einfach mit um 90° gedrehten Koordinaten und Winkeln. Aus den Abschnitten a1 und a2 auf der Y-Achse und den Steigungen s1 und s2 lässt sich anschließend der Schnittpunkt berechnen:

```
s1=tan(oldHead*2*PI/360.0);
s2=tan(heading_mou*2*PI/360.0);
a1=old_y-s1*old_x;
a2=y_mou-s2*x_pos;
xd=(a2-a1)/(s1-s2);
yd=(s1*a2-s2*a1)/(s1-s2);
radius=sqrt((x_mou-xd)*(x_mou-xd)+(y_mou-yd)*(y_mou-yd));
```

```
if (dHead>0) {
radius=-radius;
}
```

Das Vorzeichen von `radius` gibt an, ob der Mittelpunkt links (negative Werte) oder rechts (positive Werte) vom Zentrum des Bots liegt. Mit bekanntem Zentrum ergibt sich der Radius der Kreise, den die beiden Räder beschreiben, zu:

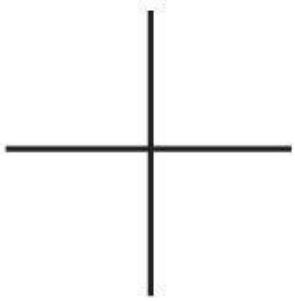
```
right_radius=radius-WHEEL_DISTANCE;
left_radius=radius+WHEEL_DISTANCE;
```

Für die Geschwindigkeiten fehlen nur noch ein wenig einfache Geometrie und die Zeitauflösung von 0,5 Sekunden:

```
v_mou_left=lastHead/360*2*PI* right_radius/0.5;
v_mou_right=lastHead/360*2*PI* left_radius/0.5;
```

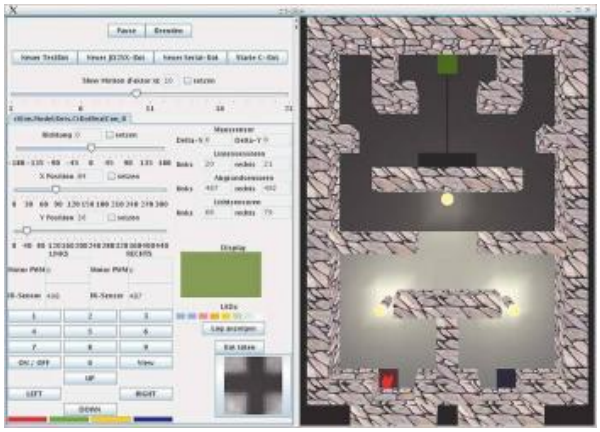
Zusammen, was zusammen gehört

Sowohl die Messdaten des Maussensors als auch die der Rad-Encoder sind nicht frei von Fehlern. Den Maussensor irritieren zum Beispiel sehr glatte oder rote Oberflächen. Die Rad-Encoder messen auch dann Bewegungen, wenn der Bot an einem Hindernis festhängt, die Räder aber aufgrund eines glatten Untergrunds durchdrehen. Daher bietet es sich an, die Messwerte zu koppeln. Dazu vergleicht man korrespondierende Wertepaare miteinander und verrechnet sie zu einem gekoppelten Wert. Im einfachsten Fall geschieht dies durch Addition der mit einem Gewichtungsfaktor multiplizierten Werte.



Ein einfaches, mit höchstmöglicher Auflösung gedrucktes Kreuz dient als Motiv, um den Maussensor zu fokussieren. Dabei sollten die Linien so schmal wie möglich sein.

Als Beispiel soll hier der Wert für die Geschwindigkeit des linken Rades dienen. Vor der Verrechnung lohnt eine Prüfung, ob die Einzelwerte nicht sehr weit auseinander liegen. Ergibt die Messung des Maussensors zum Beispiel den Wert 0 für v_mou_left, die der Rad-Encoder aber 50 für v_enc_left, so hängt der bot wohl an einem Hindernis fest. Der resultierende Wert v_left sollte dann den wahrscheinlicheren Wert - also 0 - enthalten. Da die Geschwindigkeit, mit der die Räder sich dann noch drehen, stark vom Untergrund abhängt, legt die Konstante STUCK_DIFF fest, um wie viel sich die beiden Geschwindigkeiten maximal unterscheiden dürfen.



Der c't-Sim kann das Bild des Maussensors darstellen. Ist dieses scharf, so stimmt die Fokussierung.

```
if (abs(v_enc_left-v_mou_left) < STUCK_DIFF)
v_left=(1.0-G_SPEED)*v_enc_left+G_SPEED*v_mou_left;
else
v_left=v_mou_left;
```

Die Gewichtung steuern die beiden Faktoren `G_POS` für die Positionsdaten und `G_SPEED` für alle Geschwindigkeiten. Sie sind in der Datei bot-local.h definiert und dürfen Werte von 0.0 bis 1.0 annehmen. Sie verschieben die Gewichtung linear von den Rad-Encodern (0.0) hin zum Maussensor (1.0). Ein Wert von 0.5 besagt, dass die Werte beider Sensoren gleich stark in den gekoppelten Messwert einfließen.

Maßarbeit

Litt das Verhalten zum Drehen des Roboters auf dem realen Bot stark unter den ungenauen Werten der Encoder, so erlauben die Daten des Maussenors präzise Drehungen. Schon bei der Initialisierung der Werte für das `bot_turn_behaviour()` in der zugehörigen Botenfunktion erkennt man erste Vereinfachungen durch die Verwendung des berechneten Winkels aus den Messungen des Maussensors:

```
target_angle=heading+degrees;
```

In der bisherigen Variante mussten hier erst die notwendigen Encoder-Schritte berechnet und um den aktuellen Stand der Encoder berichtigt werden:

```
turn_targetR=(degrees*ANGLE_CONSTANT)/360.0;
turn_targetL=-turn_targetR;
turn_targetR+=sensEncR;
turn_targetL+=sensEncL;
```

Das Verhalten stellte bislang anhand der beiden Ziel-Encoder-Stände turn_targetR und turn_targetL fest, ob die Drehung vollendet war und leitete dann ein Bremsmanöver und Korrekturen ein:

```
if (to_turnL <= 2 && to_turnR<=2) {
turnState=SHORT_REVERSE;
..
}
```

Mit den genaueren und laufend aktualisierten Messwerten reicht es, die Drehung zu stoppen, wenn der Bot bis auf ein Grad das Ziel erreicht hat. Die Massenträgheit des Bots dreht ihn dann zu Ende:

```
if (to_turn<2){
turnState=FULL_STOP;
...
}
```

Weitere Zustände, wie sie im bisherigen bot_turn_behaviour() für Korrekturen sorgten, fallen weg. Neben der Verbesserung bereits vorhandener Verhalten stehen nun auch völlig neue Möglichkeiten offen.

Kommandosache

Da der Bot nun seine Position kennt, kann er einfache Pläne schmieden wie: Fahre zuerst nach A, dann nach B. Um die Ausführung kümmert sich bot_gotoxy(). Dieses Verhalten erlaubt es, den Bot zu einer beliebigen Position zu schicken, die er in Form von X- und Y-Koordinaten erhält:

```
void bot_gotoxy(Behaviour_t *caller, float x, float y){
target_x=x;
target_y=y;
switch_to_behaviour(caller, bot_gotoxy_behaviour, NOOVERRIDE);
}
```

Zuerst richtet das Verhalten den Bot so aus, dass er zum Zielpunkt schaut. `bot_turn()` übernimmt das:

```
float newHeading= atan(yDiff/xDiff)*360.0/(2.0*PI);
bot_turn(data,(int16)(newHeading-heading));
```

Abweichungen von der Zielrichtung korrigiert `bot_gotoxy_behaviour()` sofort, indem es den Zielwinkel bei jedem Durchlauf neu berechnet. Ist der Abstand zum Zielpunkt kleiner als einen halben Zentimeter, stoppt das Verhalten, sodass der Bot nahezu exakt auf dem Zielpunkt zum Stehen kommt. Diese neuen Möglichkeiten bieten einigen Spielraum für zukünftige Verhalten. Führt man eine Liste der bisher angefahrenen Positionen, so kann der Bot leicht wieder zur Basis zurückkehren. Er muss dazu nur `bot_gotoxy()` mit den einzelnen Punkten der Liste in umgekehrter Reihenfolge füttern.

Bei der Genauigkeit der Einzelmessungen ist ebenfalls noch Spielraum für Optimierungen. So könnte der Bot über den SQUAL-Wert (siehe Kasten) die Qualität der Messergebnisse des Maussensors ermittelt und die Gewichtung der gekoppelten Messwerte selbstständig anpassen. (**bbe[1]**)

Literatur

[1] **Webseite zum c't-Bot-Projekt[2]**

[2] **Benjamin Benz, Carl Thiede, Thorsten Thiele, Hallo Welt!, Aufbau und Inbetriebnahme des c't-Bot[3]**

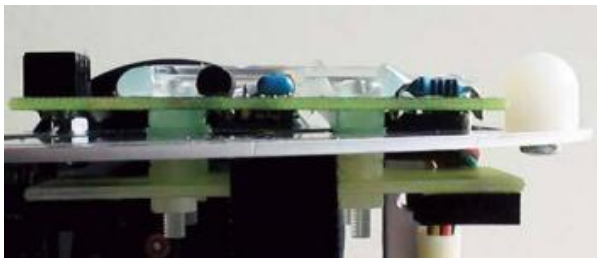
[3] **Benjamin Benz, Peter König, Lasse Schwarten, Drängelnde Spielgefährten, Kollisionen und Sensoren für den c't-Sim, neues Verhalten für den c't-Bot[4]**

[4] **Benjamin Benz, Nervensystem, Programmierung des c't-Bot von der Pike auf[5]**

Umbauten und Optimierung

Die bisherige Anordnung des Maussensors bietet zwar eine recht große Bodenfreiheit,

beeinträchtigt aber stark die Genauigkeit der Messergebnisse, da die Linse des Sensors zu weit vom Boden entfernt ist. Wie gut der eigene c't-Bot den Boden sieht, kann man mit einem einfachen Hilfsmittel feststellen. Über den c't-Sim und den Bot-2-USB-Adapter kann man sich die Bilder direkt ansehen. Als Testunterlage verwendet man beispielsweise ein Linienkreuz, gedruckt von einem guten Laser- oder Tintenstrahldrucker. Auch bei optimaler Fokussierung hat der Sensor nicht immer genau die im Datenblatt [1] angegebenen 400 CPI (Counts Per Inch). Schuld an der schwankenden Auflösung sind unter anderem die unterschiedliche Beschaffenheit des Untergrundes, Beleuchtungsunterschiede und die Qualität der Fokussierung der Sensorkamera.



Nach dem Umbau liegt das Maussandwich nahezu parallel zum Boden und damit nicht mehr parallel zur Grundplatte.

Ob ein Untergrund „gut“ oder „schlecht“ für den Sensor ist, lässt sich mit bloßem Auge nicht sehen. So bietet beispielsweise normales, glänzendes Isolierband, wie man es in jedem Baumarkt bekommt, dem Maussensor kaum Strukturen, an denen er sich orientieren kann. Für die CNY70-Sensoren, mit deren Hilfe der Bot einer Linie folgen kann, wäre das Material hingegen ideal, da es kaum Licht reflektiert. Um die Oberflächeneigenschaften, wie sie sich für den Sensor darstellen, beurteilen zu können, stellt der ADNS-2610 den Squal-Wert (Surface Quality) zur Verfügung. Er zeigt an, wie viele charakteristische Eigenschaften der Untergrund, den der Sensor gerade fotografiert, besitzt. Da der Chip aus der Bewegung dieser Eigenschaften die Änderung der X- und Y-Koordinate berechnet, sind diese auch für die Genauigkeit maßgebend. Je höher der angezeigte Wert, desto besser ist die Messgenauigkeit.

Die aktuelle Software des c't-Bot gibt ihn zusammen mit anderen Werten auf dem Display (Screen 4) aus. Als besonders gut in Bezug auf den Squal-Wert haben sich einfache Spanplatten erwiesen.

Möchte man die Genauigkeit des Sensors verbessern, muss man den Aufbau des Maus-Sandwiches verändern: Die Platinenstreifen, die bisher die Linsenplatte an ihrem Platz hielten, fallen weg. Heißkleber an jeder Ecke der Linsenplatte hält diese stattdessen an ihrem Platz. Allerdings sollte man darauf achten, dass kein Kleber auf die empfindliche Linsenplatte gerät. Muttern und Unterlegscheiben verteilt man so, dass der Sensor parallel zum Boden liegt. Die vorderen beiden Schrauben erhalten je eine Mutter sowie zwei Unterlegscheiben, die hinteren nur je eine Unterlegscheibe. Achtung: Die Basisplatte des Bots ist nicht parallel zum Untergrund. Durch diese Modifikation schrumpft die Distanz zwischen Boden und Maussensor auf die im Datenblatt empfohlenen 2,5 mm. Dies sorgt für eine gute Fokussierung, reduziert jedoch die Bodenfreiheit dramatisch. Auf Teppichböden oder Untergründen mit größeren Erhebungen wie Leisten und Fugen bleibt der Bot stecken. Ein genaue Anleitung für die Modifikation findet sich auf der Projektseite [1].

Einstellungssache

Selbst bei optimal fokussierter Kamera muss man die Messwerte noch nachbearbeiten. Dies erfolgt für Rotation und Translation getrennt in zwei Etappen. Dreht man den Roboter einmal auf der Stelle um 360 Grad, erhält man den Maussensorwert (X) für einen Vollkreis. Wer möchte, kann

sich dafür aus einer alten CD-Spindel eine Halterung basteln, die eine seitliche Abweichung verhindert und so eine exakte Drehung erlaubt. Mit den nominellen 400 CPI aus dem Datenblatt und 15 mm Abstand zwischen Sensor und Zentrum des Bots ergibt sich für eine 360°-Drehung (2 · π · 15 / 25.4) · 400 = 1484.

Dieser Wert weicht normalerweise vom abgelesenen Wert ab. Letzteren trägt man in der Datei bot-local.h ein:

Um die Optik des Maus-sensors besser zu fokussieren, entfernt man die beiden kleinen Platinenstreifen und fixiert die Linsen-platte vorsichtig mit Heißkleber.

```
#define MOUSE_FULL_TURN 1510
```

Wie in diesem Beispiel zu sehen, kann der Wert auch durchaus über dem Nominalwert liegen, was einer CPI-Auflösung von mehr als 400 entspricht.

Für die Kalibrierung der Y-Komponente, die für Streckenmessungen verantwortlich ist, schiebt man den Roboter einen Meter vorwärts. Auch hier zeigt das Display einen Wert an, der sich in eine Auflösung umrechnen lässt. Für einen gemessenen Wert von 15 100 ergibt sich ein CPI-Wert von 384 (15 100 / 100 · 2,54), der ebenfalls in der Datei bot-local.h festgehalten wird:

```
#define MOUSE_CPI 384
```

Die Qualität der Kalibrierung hängt nicht nur von präzisen Bewegungen des Bots ab, sondern auch von einem Untergrund, auf dem der Maussensor genug Merkmale erkennt.

URL dieses Artikels:
http://www.heise.de/-290526

Links in diesem Artikel:
[1] mailto:bbe@ct.de
[2] https://www.heise.de/ct/artikel/c-t-Bot-und-c-t-Sim-284119.html
[3] https://www.heise.de/ct/artikel/Hallo-Welt-290314.html

[4] https://www.heise.de/ct/artikel/Draengelnde-Spielgefaehrten-290334.html

[5] https://www.heise.de/ct/artikel/Nervensystem-290376.html