Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Fakultät: Elektrotechnik und Informatik
Studiengang: Angewandte Informatik

Dokumentation Robotik Praktikum

Erstgutachter: Prof. Dr.-Ing Konrad Wöllhaf

Vorgelegt von: Christian Högerle, 24966

Thomas Buck, 24956

Fach: Robotik Praktikum

Eingereicht am: 17.01.2017

Inhaltsverzeichnis

Aufgabe 1: 3
Aufgabe 2: 5
Aufgabe 3: 7
Aufgabe 4: 10
Aufgabe 5: 11

Aufgabe 1:

Aufgabenstellung:

Ziel ist, dass die Roboterhand einer Kartonbox folgt.

Lösung der Aufgabe:

Zunächst muss der TCP (Tool Center Point) der Roboterhand geteached werden. Dafür wurde die Spitze des Werkzeugs von vier verschiedenen Seiten an einen Punkt geführt. Dadurch der kann der Roboterarm im Tool-Koordinatensystem beweget werden, wodurch das Werkzeug komfortabel im Raum bewegt werden kann.

Anschließend wurde ein Basis-Koordinatensystem auf der Oberfläche des Kartons platziert und in den Roboterarm geteached. Die Achsen des Koordinatensystems liegen an den Kanten der Box, um sich einfach an diesen entlang bewegen zu können.

Zuletzt haben wir ein kleines Beispielprogramm erstellt, welches den Kanten der Box folgt. Dafür verwendeten wir PTP (Point-to-Point), LIN (lineare Bewegung) und CIR (kreisförmige Bewegung) Befehle.
Zur Veranschaulichung des Basis-Koordinatensystems wurde die Box verschoben. Anstatt sämtliche Punkte neu zu bestimmen, musste nur die neue Basis geteached werden.

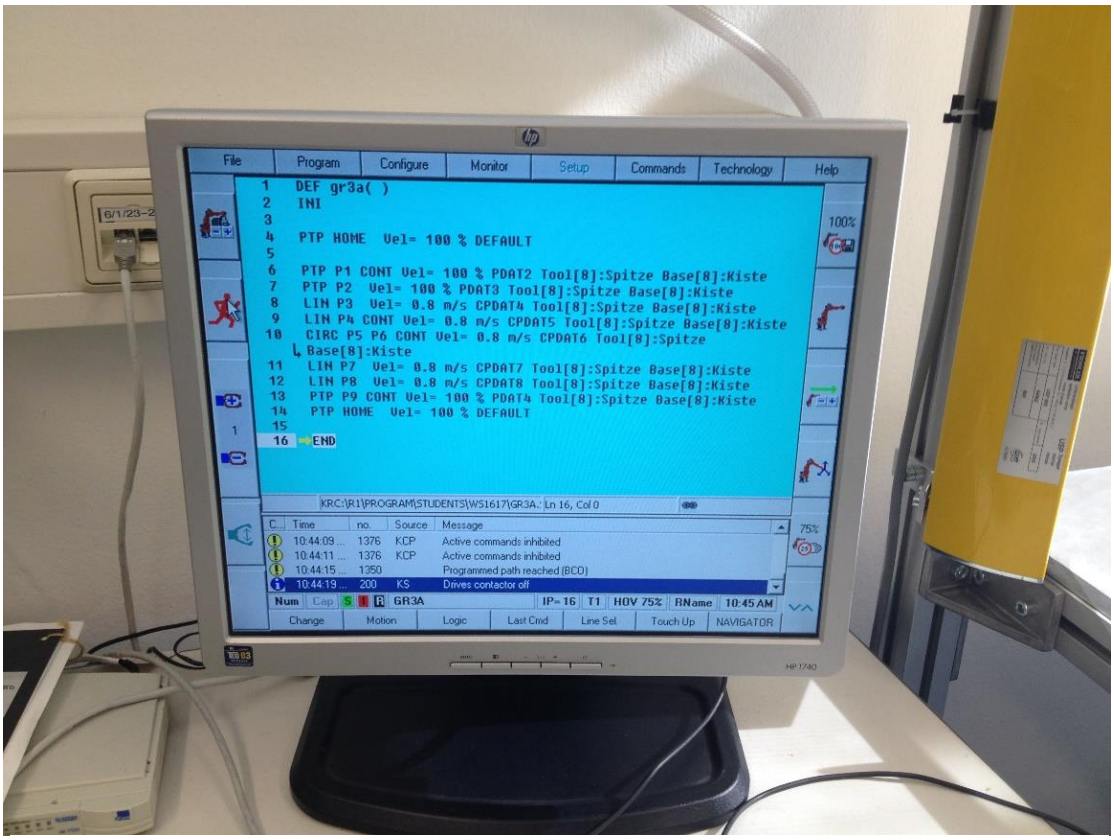


Abbildung 1: Programmcode Aufgabe 1

Abbildung 2: Roboter in kreisförmiger Bewegung

Aufgabe 2:

Ziel der Aufgabe:

Ziel dieser Aufgabe ist es, dass die Roboterhand einer quadratischen Fläche folgt. Dazu soll der Roboter eine Kante nicht mit einer kontinuierlichen Bewegung folgen, sondern kleine inkrementelle Schritte ausführen, um den Punktschweißvorgang zu simulieren.

Lösung der Aufgabe:

Für diese Aufgabe wurde die Kuka Simulation verwendet. Zunächst wurden der TCP und die Basis per numerischer Eingabe definiert, das manuelle Teachen entfällt dadurch. Anschließend wurden die vier Eckpunkte der Fläche festgelegt. Die drei linearen Bewegungen konnten wir von der vorherigen Aufgabe übernehmen. Für die inkrementelle Bewegung wurde eine 3D-Punkt Variable angelegt und mit dem Startpunkt der Bewegung initialisiert. In einer Schleife konnten nun die Y-Koordinate wiederholt erhöht und der neue Punkt angefahren werden.

Wir hatten große Schwierigkeiten die Simulation zu starten. Durch die Windows Firewall hatten wir Probleme bei der Kommunikation zwischen Simulationssoftware und Roboteremulation. Dennoch bietet die Simulation im Vergleich zum realen Roboter einen deutlichen Vorteil, da beim Testen keine teuren Geräte beschädigt werden können.

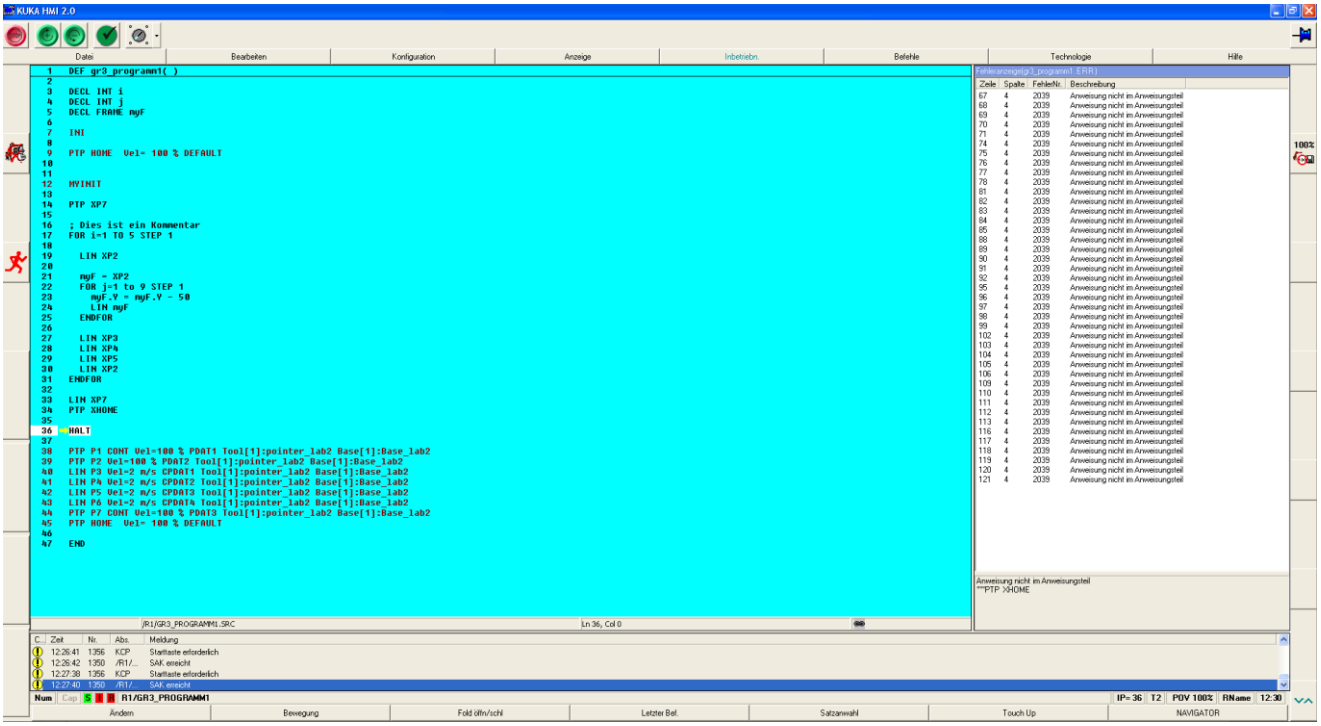


Abbildung 3: Programmcode in der Roboteremulation

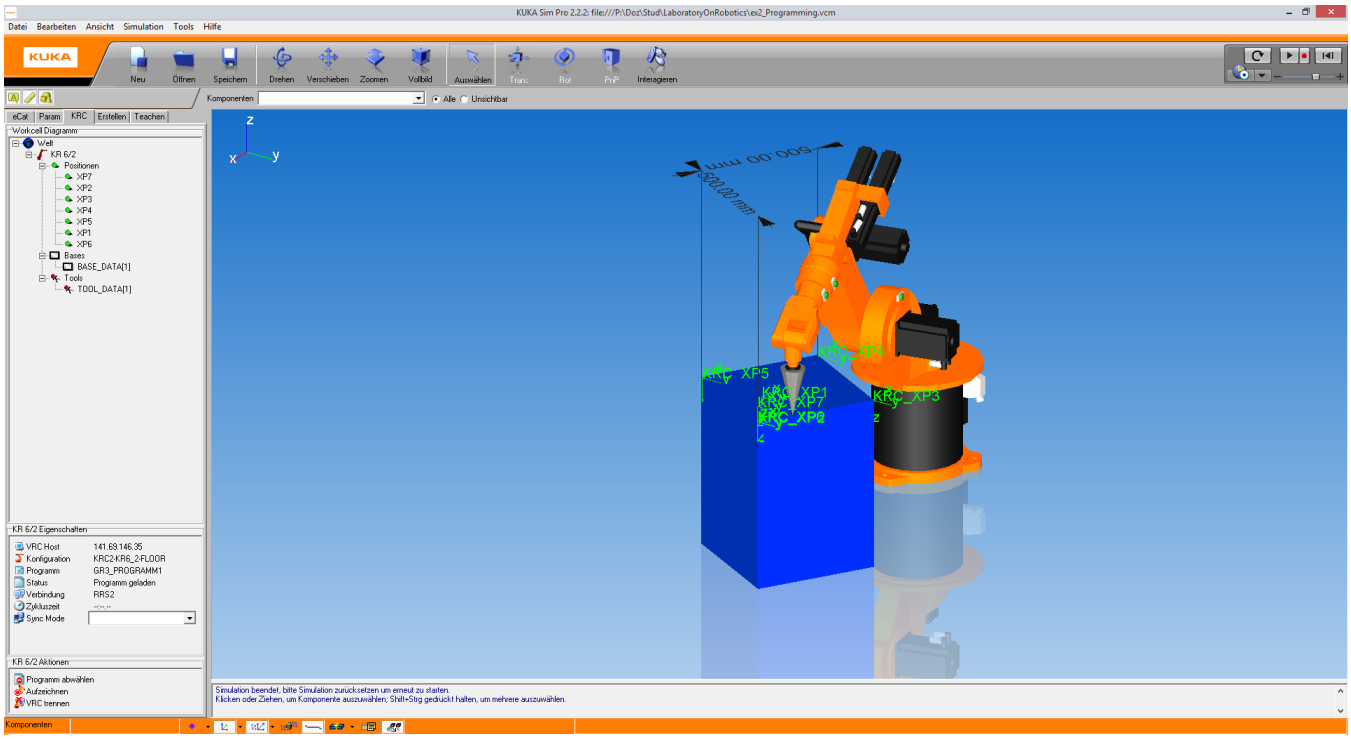


Abbildung 4: Roboter in der Simulation

Aufgabe 3:

Ziel der Aufgabe:

Um den Roboter herum wurden verschiedene Geräte kreisförmig aufgestellt:

- Ein sogenannter Feeder gibt neue Tennisbälle aus und teilt dem Roboter über ein digitales Signal mit, ob noch weitere Bälle zur Verfügung stehen.
- Eine Waage mit zwei Waagschalen mit verschiedenen Druckpunkten. Ein Ball muss nacheinander auf beide Schalen gelegt werden, über zwei digitale Signale kann während dem Vorgang ausgelesen werden, ob die Waage ausgelöst wurde.
- Drei Paletten mit je 12 Slots für Tennisbälle. Die verschiedenen Bälle sollen nach Gewicht in die Paletten sortiert werden.

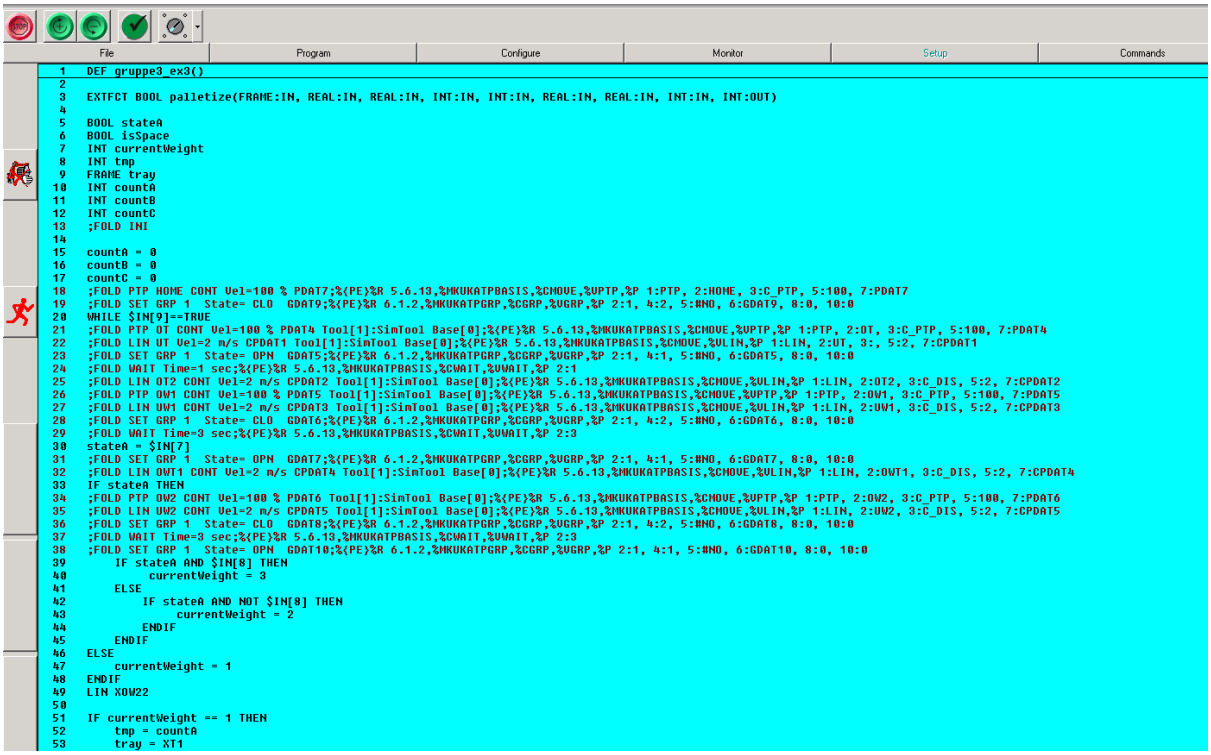
Lösung der Aufgabe:

Uns wurde ein simulierter Aufbau zur Verfügung gestellt. Dieser enthielt bereits die notwendigen Punkte um die verschiedenen Geräte anfahren zu können. Unsere Aufgabe bestand darin, die Logik des Programms zu implementieren.

Zuerst wird ein neuer Ball aus dem Feeder gegriffen und zur ersten Waagschale bewegt. Während dem Wiegevorgang muss das Greifwerkzeug geöffnet sein und mehrere Sekunden lang gewartet werden. Wenn die erste Waage bereits erkennt, dass der Ball sicher zu einer der drei Kategorien gehört, wird der zweite Wiegevorgang übersprungen. Ansonsten wird der Ball auf die zweite Schale gelegt.

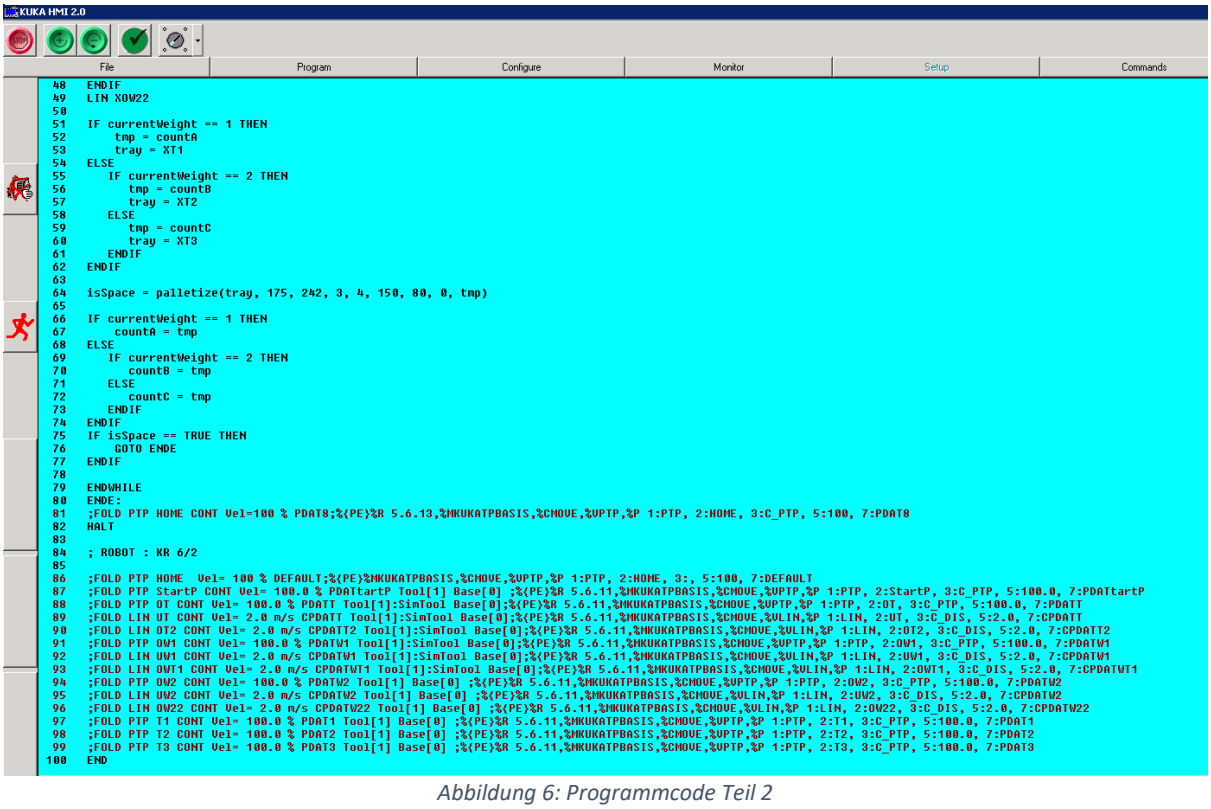
Nun ist das Gewicht des aktuellen Balls bekannt und er muss einsortiert werden. Dafür stand uns die bereits implementierte Hilfsfunktion „palletize()“ zur Verfügung. Diese fährt die, per Parameter angegebene, Palette an der nächsten freien Position an und legt den Ball ab.

Dieser Arbeitsablauf wird wiederholt solange noch weitere Bälle im Feeder zur Verfügung stehen.



```
DEF gruppe3_ex3()

EXTFCT BOOL palletize(FRAME:IN, REAL:IN, REAL:IN, INT:IN, INT:IN, REAL:IN, REAL:IN, INT:IN, INT:OUT)

BOOL stateA
BOOL isSpace
INT currentWeight
INT tmp
FRAME tray
INT countA
INT countB
INT countC
;FOLD INI

countA = 0
countB = 0
countC = 0
;FOLD PTP HOME CONT Vel=100 % PDAT7;%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:C_PTP, 5:100, 7:PDAT7
;FOLD SET GRP 1  State= CLO  GDAT9;%{PE}%R 6.1.2,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:2, 5:#NO, 6:GDAT9, 8:0, 10:0
WHILE $IN[9]==TRUE
;FOLD PTP OT CONT Vel=100 % PDAT4 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:OT, 3:C_PTP, 5:100, 7:PDAT4
;FOLD LIN UT Vel=2 m/s CPDAT1 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:UT, 3:, 5:2, 7:CPDAT1
;FOLD SET GRP 1  State= OPN  GDAT5;%{PE}%R 6.1.2,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:1, 5:#NO, 6:GDAT5, 8:0, 10:0
;FOLD WAIT Time=1 sec;%{PE}%R 5.6.13,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 2:1
;FOLD LIN OT2 CONT Vel=2 m/s CPDAT2 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:OT2, 3:C_DIS, 5:2, 7:CPDAT2
;FOLD PTP OW1 CONT Vel=100 % PDAT5 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:OW1, 3:C_PTP, 5:100, 7:PDAT5
;FOLD LIN UW1 CONT Vel=2 m/s CPDAT3 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:UW1, 3:C_DIS, 5:2, 7:CPDAT3
;FOLD SET GRP 1  State= CLO  GDAT6;%{PE}%R 6.1.2,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:2, 5:#NO, 6:GDAT6, 8:0, 10:0
;FOLD WAIT Time=3 sec;%{PE}%R 5.6.13,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 2:3
stateA = $IN[7]
;FOLD SET GRP 1  State= OPN  GDAT7;%{PE}%R 6.1.2,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:1, 5:#NO, 6:GDAT7, 8:0, 10:0
;FOLD LIN OWT1 CONT Vel=2 m/s CPDAT4 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:OWT1, 3:C_DIS, 5:2, 7:CPDAT4
IF stateA THEN
;FOLD PTP OW2 CONT Vel=100 % PDAT6 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:OW2, 3:C_PTP, 5:100, 7:PDAT6
;FOLD LIN UW2 CONT Vel=2 m/s CPDAT5 Tool[1]:SimTool Base[0];%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:UW2, 3:C_DIS, 5:2, 7:CPDAT5
;FOLD SET GRP 1  State= CLO  GDAT8;%{PE}%R 6.1.2,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:2, 5:#NO, 6:GDAT8, 8:0, 10:0
;FOLD WAIT Time=3 sec;%{PE}%R 5.6.13,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 2:3
;FOLD SET GRP 1  State= OPN  GDAT10;%{PE}%R 6.1.2,%MKUKATPGRP,%CGRP,%VGRP,%P 2:1, 4:1, 5:#NO, 6:GDAT10, 8:0, 10:0
    IF stateA AND $IN[8] THEN
        currentWeight = 3
    ELSE
        IF stateA AND NOT $IN[8] THEN
            currentWeight = 2
        ENDIF
    ENDIF
ELSE
    currentWeight = 1
ENDIF
LIN XOW22

IF currentWeight == 1 THEN
    tmp = countA
    tray = XT1
```

Abbildung 5: Programmcode Teil 1



```
ENDIF
LIN XOW22

IF currentWeight == 1 THEN
    tmp = countA
    tray = XT1
ELSE
   IF currentWeight == 2 THEN
      tmp = countB
      tray = XT2
   ELSE
      tmp = countC
      tray = XT3
   ENDIF
ENDIF

isSpace = palletize(tray, 175, 242, 3, 4, 150, 80, 0, tmp)

IF currentWeight == 1 THEN
    countA = tmp
ELSE
   IF currentWeight == 2 THEN
      countB = tmp
   ELSE
      countC = tmp
   ENDIF
ENDIF
IF isSpace == TRUE THEN
   GOTO ENDE
ENDIF

ENDWHILE
ENDE:
;FOLD PTP HOME CONT Vel=100 % PDAT8;%{PE}%R 5.6.13,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:C_PTP, 5:100, 7:PDAT8
HALT

; ROBOT : KR 6/2

;FOLD PTP HOME  Vel= 100 % DEFAULT;%{PE}%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:, 5:100, 7:DEFAULT
;FOLD PTP StartP CONT Vel= 100.0 % PDATtartP Tool[1] Base[0] ;%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:StartP, 3:C_PTP, 5:100.0, 7:PDATtartP
;FOLD PTP OT CONT Vel= 100.0 % PDATT Tool[1]:SimTool Base[0];%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:OT, 3:C_PTP, 5:100.0, 7:PDATT
;FOLD LIN UT CONT Vel= 2.0 m/s CPDATT Tool[1]:SimTool Base[0];%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:UT, 3:C_DIS, 5:2.0, 7:CPDATT
;FOLD LIN OT2 CONT Vel= 2.0 m/s CPDATT2 Tool[1]:SimTool Base[0];%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:OT2, 3:C_DIS, 5:2.0, 7:CPDATT2
;FOLD PTP OW1 CONT Vel= 100.0 % PDATW1 Tool[1]:SimTool Base[0];%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:OW1, 3:C_PTP, 5:100.0, 7:PDATW1
;FOLD LIN UW1 CONT Vel= 2.0 m/s CPDATW1 Tool[1]:SimTool Base[0];%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:UW1, 3:C_DIS, 5:2.0, 7:CPDATW1
;FOLD LIN OWT1 CONT Vel= 2.0 m/s CPDATWT1 Tool[1]:SimTool Base[0];%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:OWT1, 3:C_DIS, 5:2.0, 7:CPDATWT1
;FOLD PTP OW2 CONT Vel= 100.0 % PDATW2 Tool[1] Base[0] ;%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:OW2, 3:C_PTP, 5:100.0, 7:PDATW2
;FOLD LIN UW2 CONT Vel= 2.0 m/s CPDATW2 Tool[1] Base[0] ;%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:UW2, 3:C_DIS, 5:2.0, 7:CPDATW2
;FOLD LIN OW22 CONT Vel= 2.0 m/s CPDATW22 Tool[1] Base[0] ;%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:OW22, 3:C_DIS, 5:2.0, 7:CPDATW22
;FOLD PTP T1 CONT Vel= 100.0 % PDAT1 Tool[1] Base[0] ;%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:T1, 3:C_PTP, 5:100.0, 7:PDAT1
;FOLD PTP T2 CONT Vel= 100.0 % PDAT2 Tool[1] Base[0] ;%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:T2, 3:C_PTP, 5:100.0, 7:PDAT2
;FOLD PTP T3 CONT Vel= 100.0 % PDAT3 Tool[1] Base[0] ;%{PE}%R 5.6.11,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:T3, 3:C_PTP, 5:100.0, 7:PDAT3
END
```

Abbildung 6: Programmcode Teil 2

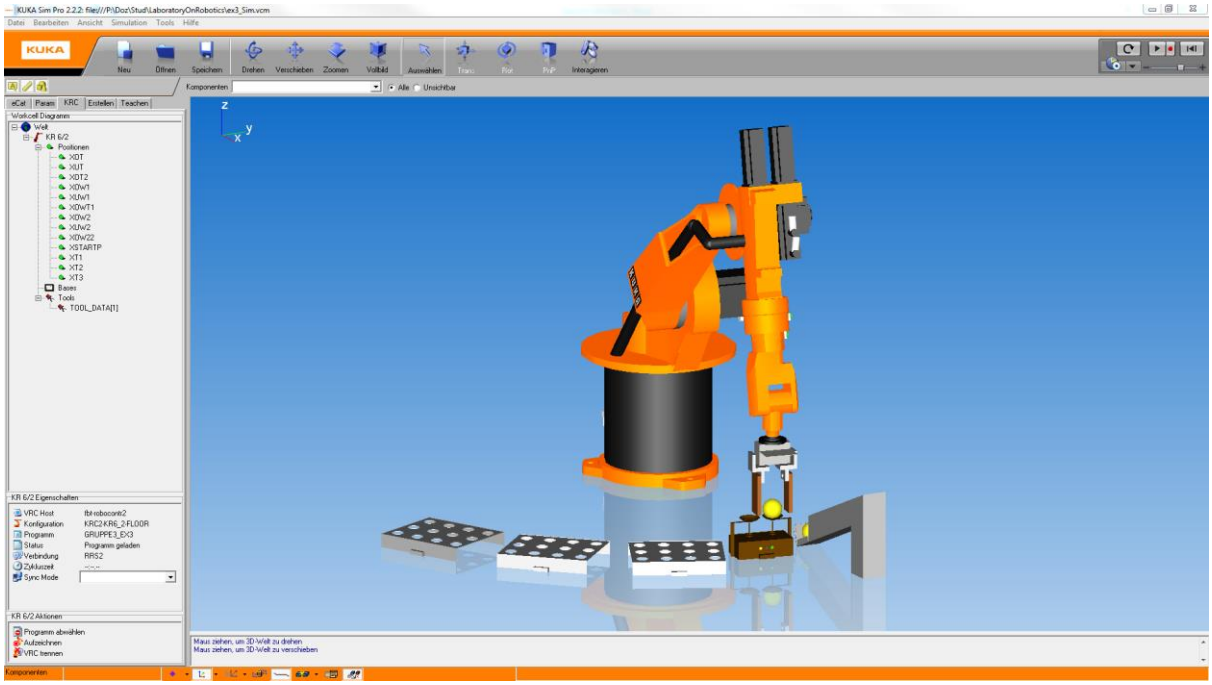


Abbildung 7: Ansicht Simulation

Aufgabe 4:

Ziel der Aufgabe:

Das simulierte Programm aus Aufgabe 3 soll nun am realen Roboter getestet werden.

Lösung der Aufgabe:

Um keine Anpassungen am Programm durchführen zu müssen werden in unserem Test die Geräte (Feeder, Waage, Paletten) gemäß des Aufbaus der Simulation platziert. Im realen Umfeld würde man die Basen der einzelnen Geräte neu bestimmen, da die Geräte nicht unbedingt frei platziert werden können.

Bei den ersten Tests mit dem realen Roboter wurden noch kleinere Probleme festgestellt. So war zum Beispiel die Wartezeit während dem Wiegen, vor der Wiederaufnahme des Balls zu kurz, was bei höheren Geschwindigkeiten zu Fehlmessungen führte. Durch die korrekte Anordnung der Wartezeiten konnte dieses Problem jedoch schnell behoben werden.

Aufgabe 5:

Problemstellung:

Der ASURO soll einer schwarzen Line mit Hilfe eines PID-Reglers folgen.

Beschreibung der Steuerung:

Beim Initialisierungsvorgang werden die ADC-Werte (linker und rechter Sensor) der schwarzen Linie gespeichert. Ob sich der ASURO außerhalb der Strecke befindet, kann über diese Werte festgestellt werden. Wird das Verlassen der Strecke festgestellt, fährt der ASURO mit konstanter Geschwindigkeit rückwärts. Sobald die Sensoren die schwarze Linie erkennen, springt der ASURO wieder in die PID-Regler Routine. Zusätzlich werden bei der Initialisierung die Werte (P, I und D) des PID-Reglers gesetzt. Der PID-Regler ist in eine Bibliothek ausgelagert. Der Mikrocontroller ruft in der while-Schleife die Funktion „lineFollower“ auf. Diese Funktion ermittelt und speichert die Werte der Sensoren einmal mit ausgeschalteter LED und mit eingeschaltener LED. Durch dieses Verfahren kann die Abhängigkeit der Sensoren vom Umgebungslicht kompensiert werden. Sind die Werte ermittelt, werden die Werte dem PID-Regler übergeben. Die Funktion pidExcute (siehe Abbildung 1) berechnet den jeweiligen PID-Wert zu den übergebenen Werten. Durch das vorher beschriebene Verfahren ist der should-Wert konstant 0.

```
double pidExecute(double should, double is, PIDState *state) {
    unsigned long now = Gettime();
    double timeChange = (double)(now - state->last);
    double error = should - is;
    double newErrorSum = state->sumError + (error * timeChange);
    if ((newErrorSum >= state->intMin) && (newErrorSum <= state->intMax))
        state->sumError = newErrorSum; // Prevent Integral Windup
    double dError = (error - state->lastError) / timeChange;
    double output = (state->kp * error) + (state->ki * state->sumError) + (state->kd * dError);
    state->lastError = error;
    state->last = now;
    if (output > state->outMax) {
        output = state->outMax;
    }
    if (output < state->outMin) {
        output = state->outMin;
    }
    return output;
}

void pidSet(PIDState *pid, double kp, double ki, double kd, double min, double max, double iMin, double iMax) {
    pid->kp = kp;
    pid->ki = ki;
    pid->kd = kd;
    pid->outMin = min;
    pid->outMax = max;
    pid->intMin = iMin;
    pid->intMax = iMax;
    pid->lastError = 0;
    pid->sumError = 0;
    pid->last = 0;
}
```

Abbildung 8: PID-Regler Quellcode

Anhand des Vorzeichens des PID-Wertes kann die nötige Bewegungsrichtung ermittelt werden. Der eigentlich PID-Wert wird zur Anpassung der Motorgeschwindigkeit verwendet. Dabei wird der berechnete PID-Wert auf beide Motoren jeweils zur Hälfte aufgeteilt. Das bedeutet, dass zum Beispiel der Motor rechts um PID/2 schneller wird und Motor links um PID/2 langsamer. Nach der „lineFollower“ Funktion werden zwei Taster (K2 und K5) abgefragt. Diese Abfrage dient der Erkennung, ob der ASURO am Ende der Strecke angelangt ist. Der ASURO soll eine Drehung durchführen sobald beide Sensoren gedrückt werden.

Ablauf der Drehung:

- Motorgeschwindigkeit auf 0 setzen und bremsen
- Die aktuellen Werte der Sensoren speichern (schwarzer Bereich)
- Den linken Motor auf Drehrichtung vorwärts, Den rechten Motor auf Drehrichtung rückwärts
- Motorgeschwindigkeit: links: 112, rechts: 107 (Motoren laufen nicht gleich schnell)
- 200 ms Sekunden sleep um den schwarzen Bereich zu verlassen
- While-Schleife: Sensorwerte abfragen und bei Erkennung des schwarzen Bereichs anhalten
- While-Schleife verlassen
- Die ursprüngliche while-Schleife wird nun wieder ausgeführt.

Quellcode: https://github.com/g40st/ASURO_Line_Follower

Ermittlung der Werte P, I und D:

Die Werte wurden per trial and error festgelegt.

Probleme:

Für die Drehung sollte die Funktion GoTurn(int distance, int degree, int speed) verwendet werden. Die Funktion verwendet ein Odometer um die Distanz beziehungsweise den Winkel zu berechnen. Dabei werden die Odometrie Sensoren im Interrupt-Betrieb verwendet. Zunächst sollte eine 180 Grad Drehung mit Hilfe der Funktion vollzogen werden. Allerdings endete dies in einer Endlosschleife. Nach einer genaueren Analyse stellte sich heraus, dass die Sensoren nicht in Ordnung waren. Anhand dieses Beispielprogramms wurden die Odometire Sensoren getestet. Abbildung 2 zeigt, dass die Sensoren konstant die Werte 1 zurückgeben.

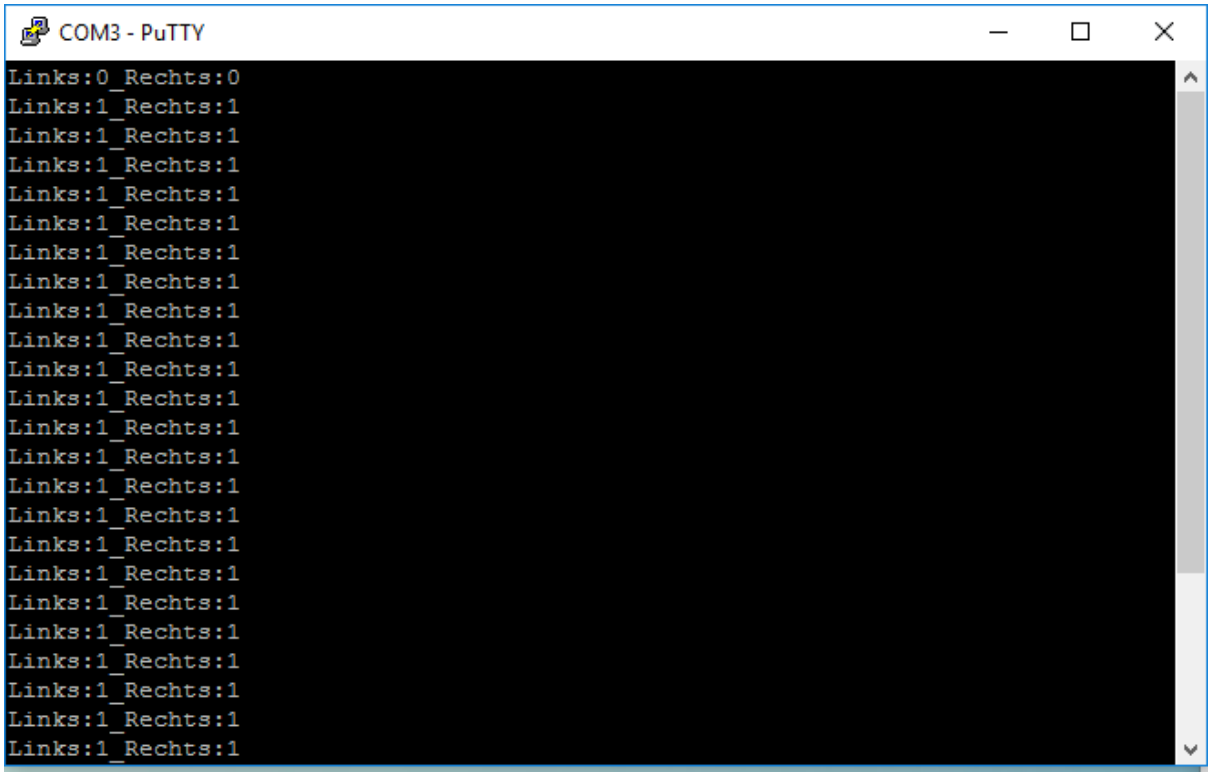
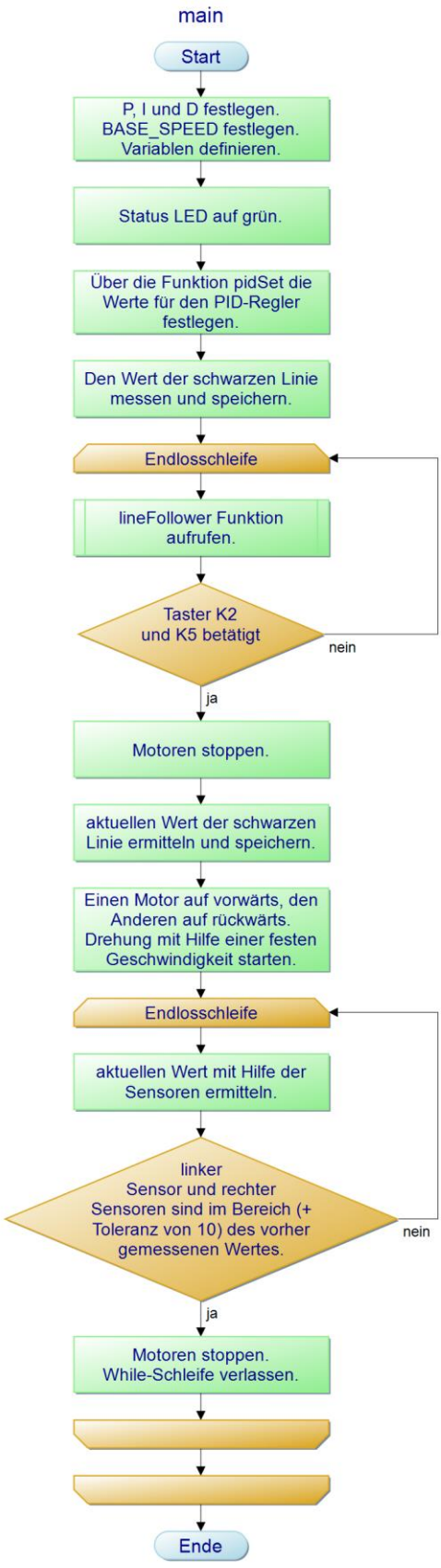
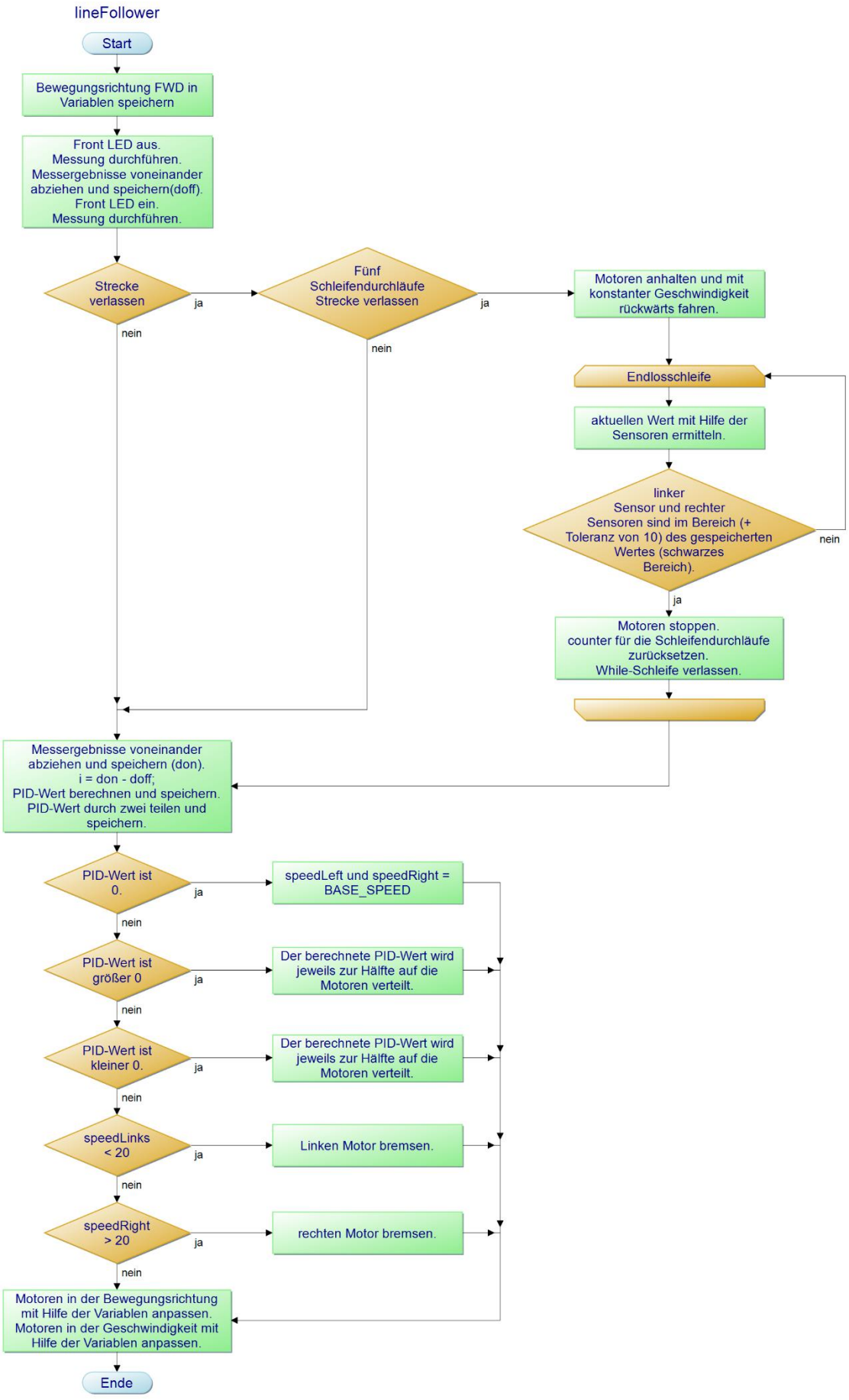


Abbildung 9: Terminalfenster Beispielprogramm

Ablaufdiagramme:



.


lineFollower
Start
Bewegungsrichtung FWD in Variablen speichern
Front LED aus.
Messung durchführen.
Messergebnisse voneinander abziehen und speichern(doff).
Front LED ein.
Messung durchführen.
Strecke verlassen
ja
nein
Fünf Schleifendurchläufe Strecke verlassen
ja
nein
Motoren anhalten und mit konstanter Geschwindigkeit rückwärts fahren.
Endlosschleife
aktuellen Wert mit Hilfe der Sensoren ermitteln.
linker Sensor und rechter Sensoren sind im Bereich (+ Toleranz von 10) des gespeicherten Wertes (schwarzes Bereich).
nein
ja
Motoren stoppen.
counter für die Schleifendurchläufe zurücksetzen.
While-Schleife verlassen.
Messergebnisse voneinander abziehen und speichern (don).
i = don - doff;
PID-Wert berechnen und speichern.
PID-Wert durch zwei teilen und speichern.
PID-Wert ist 0.
ja
speedLeft und speedRight = BASE_SPEED
nein
PID-Wert ist größer 0
ja
Der berechnete PID-Wert wird jeweils zur Hälfte auf die Motoren verteilt.
nein
PID-Wert ist kleiner 0.
ja
Der berechnete PID-Wert wird jeweils zur Hälfte auf die Motoren verteilt.
nein
speedLinks < 20
ja
Linken Motor bremsen.
nein
speedRight > 20
ja
rechten Motor bremsen.
nein
Motoren in der Bewegungsrichtung mit Hilfe der Variablen anpassen.
Motoren in der Geschwindigkeit mit Hilfe der Variablen anpassen.
Ende