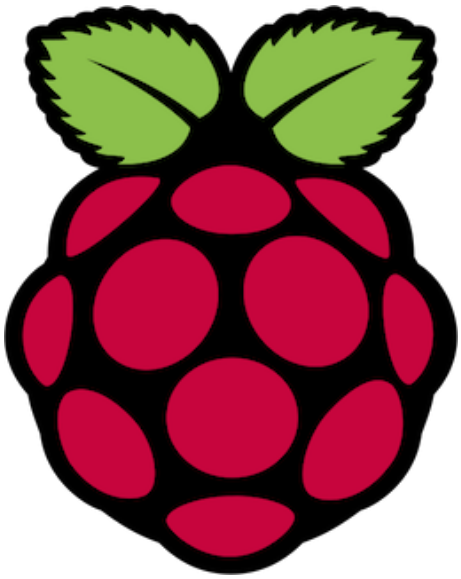
Lehrstuhl für
Kontinuumsrobotik

Leibniz
Universität
Hannover

Vorlesung/Experimentelle Übung:
Programmierung mechatronischer Systeme

Hausarbeit

Raspberry Pi Roboter (Team04)



Li Chen
Matrikelnummer 10016432

Hongyu Zhao
Matrikelnummer 10016275

Sommersemester 2018
Hannover, 27. Juli 2018

Dozentin: Prof. Dr.-Ing. J. Burgner-Kahrs
Betreuer: M.Sc. E. Amanov, Dipl.-Ing. V.Modes

Inhaltsverzeichnis

1 Einleitung

Der Raspberry Pi ist ein Einplatinencomputer, der von der britischen RaspberryPiFoundation entwickelt wurde. Der Rechner enthält ein Ein-Chip-System von Broadcom mit einem ARM-Mikroprozessor, die Grundfläche der Platine entspricht etwa den Abmessungen einer Kreditkarte.

In dieser Lehrveranstaltung, Programmierung mechatronischer Systeme, werden objektorientierte Programmiersprache, die mit verschiedenen Sensoren und Elektromotor verbunden ist und auf Raspbeery Pi 3 realisiert, unterrichtet und angewendet. Zuerst wird C++ Programmiersprache angewendet, um die Elektromotor zu kontrollieren. Dann werden das Ultraschallsensor, der Encoder, die Liniensensoren sowie das Farbsensor mit Raspberry Pi 3 sich verschaltet, um die Geschwindigkeit der Motoren besser zu kontrollieren.

Die Arbeit besteht aus hauptsächlich 4 Teile, das Hardwareteil, die softwaretechnische Umsetzung, die Diskussion sowie die Zusammenfassung. Beim Hardwareteil werden die Anwendung der verschiedenen Sensoren, die Eigenschaften des Roboters sowie die Schaltpläne beschrieben. Bei der softwaretechnische Umsetzung, die aus 3 Teile besteht, werden Programmierung für Linie flogen, Odometrie sowie endliche Logistik ausführlich erklärt. Bei der Diskussion werden die Probleme und die Herausforderungen, die uns begegnet sind, und wie die Probleme gelöst und die benutzte Methoden sowie was als nächstes gemacht werden würde, werden erklärt. Zuletzt werden was wir gelernt haben und alle Detail zum Schluss zusammengefasst.

Im Allgemeinen ist die Programmierung mechatronischer Systeme sehr gute Lehrveranstaltung, um die objektorientierte Programmiersprache, Arbeitsprinzip verschiedener Sensoren, sowie PID Regler zu kennen.

2 Der Roboter

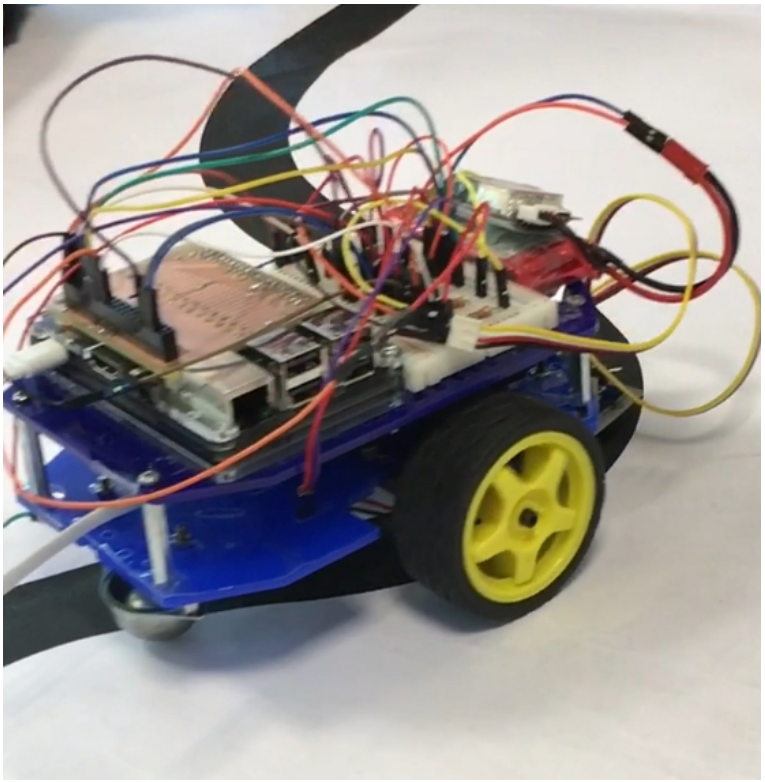


Bild 2.1 Wie das Bild gezeigt, der Roboter besteht aus Einplatinencomputer,Mobile Plattform und verschiedene Sensoren. Die Höhe des Roboters beträgt 13 cm, die Länge beträgt 18,2 cm, die Breite 11 cm und das Gewicht 0,8 kg.

2.1 Einplatinencomputer

Der Raspberry Pi 3 ist ein Computer im Scheckkartenformat, der praktisch den gleichen Funktionsumfang wie ein Desktop-PC bietet. Der Raspberry Pi bietet viele Möglichkeiten, um verschiedene Signalen zu verarbeiten und verschiedene Programmiersprache zu realisieren. Aufgrund seiner hervorragenden Grafikleitung ermöglicht der Raspberry Pi 3 Videostreaming in Blu-Ray-Qualität.Wenn den Pi in Embedded-Design-Projekt zu integrieren, er kann der 40-polige GPIO-Header mit einem Abstand von 0,1 Zoll Zugriff auf 27 GPIO-, UART-, I2C-, SPI-Leitungen sowie auf 3,3-V- und 5-V-Spannungsversorgung bieten.

2.2 Mobile Plattform

Der Raspberry Pi ist durch ein Steckbrett und eine große Anzahl von Drähten mit verschiedenen Sensoren, Widerstände und Motoren sowie andere elektronische Komponenten. Die Schaltpläne werden auf Fritzing gezeigt.

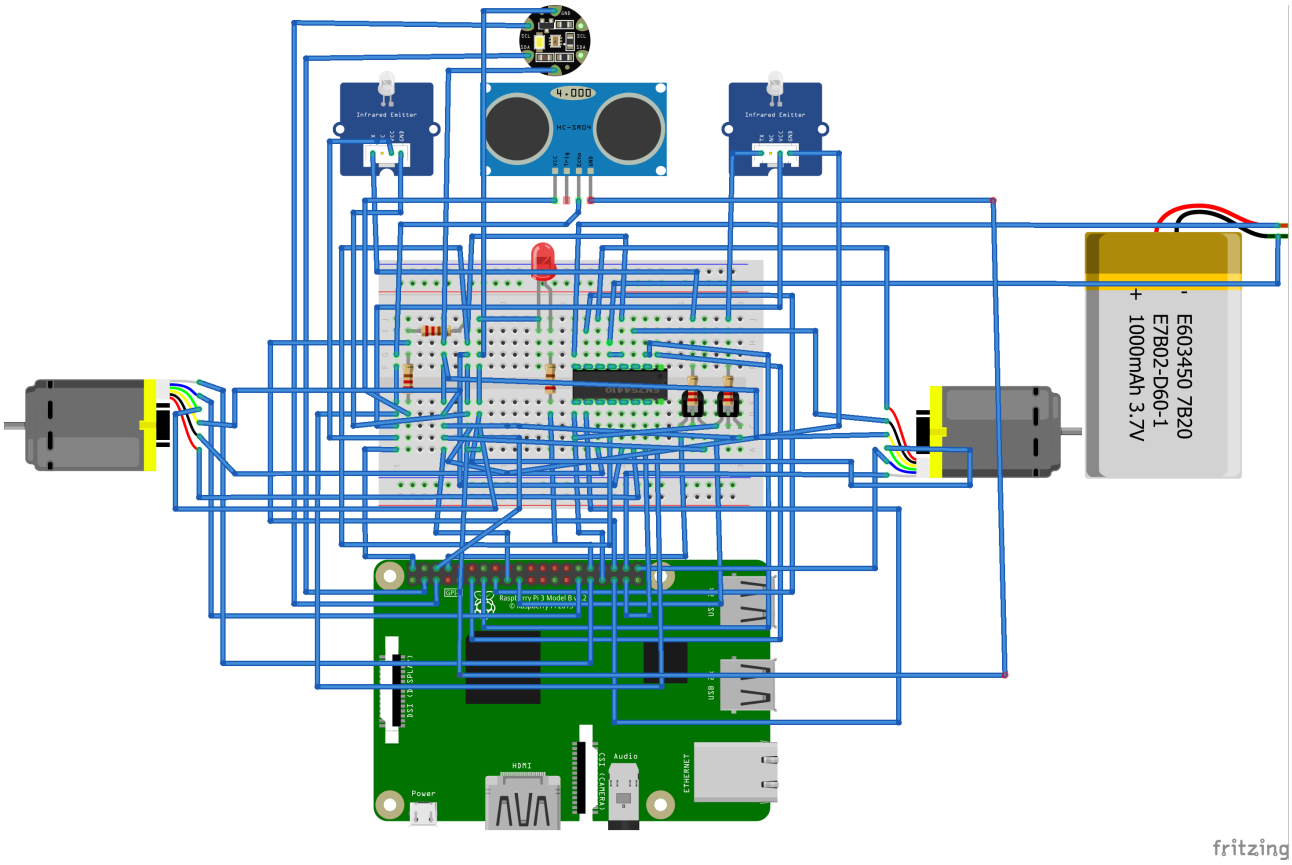


Bild 2.2 Schaltplan

2.3 Sensorik

2.3.1 Liniensensor

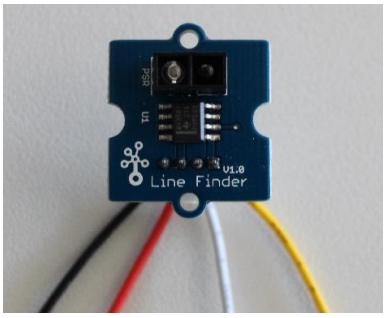


Bild 2.3Liniensensor

Das Liniensensor kann Farbe Schwarz und Weiß unterscheiden und registrieren. Das Signal “0” bedeutet Weiß. Das Signal “1” bedeutet Schwarz.

2.3.2 Encoder

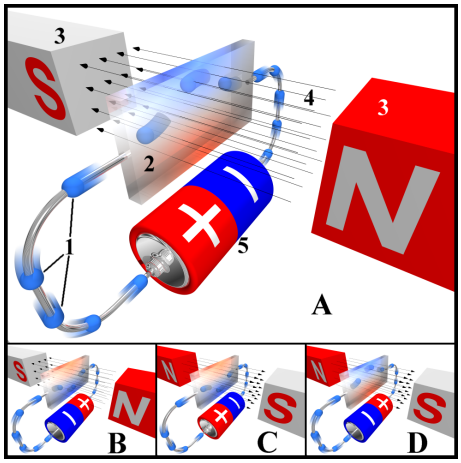


Bild 2.4 Hall Effect

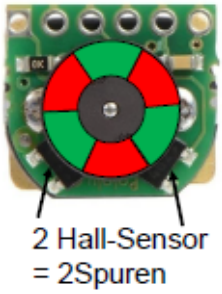


Bild 2.5 Encoder

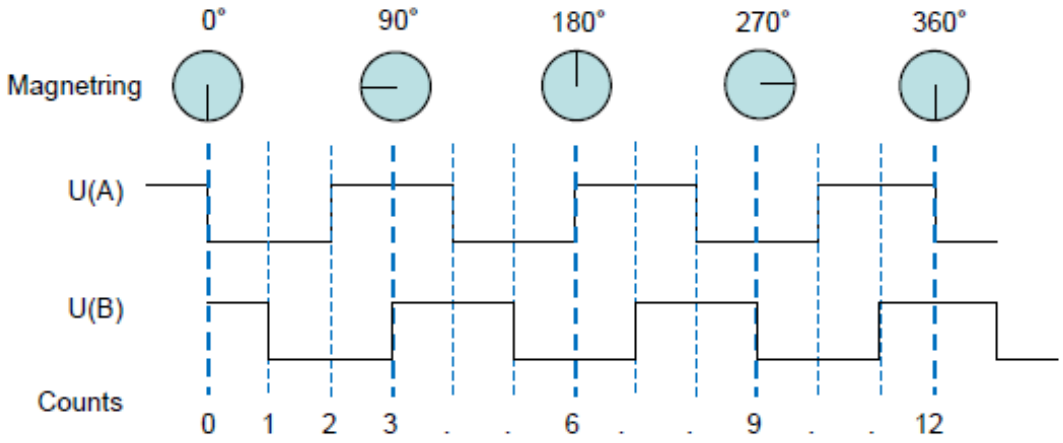


Bild 2.6

Der theoretische Grundlage des Encoder ist der Hall-Effekt. Der Hall-Effekt, benannt nach Edwin Hall, beschreibt das Auftreten einer elektrischen Spannung in einem stromdurchflossenen Leiter, der sich in einem stationären Magnetfeld befindet. Die Spannung fällt dabei senkrecht sowohl zur Stromfluss- als auch zur Magnetfeldrichtung am Leiter ab und wird Hall-Spannung genannt.

Das Bild zeigt, wie der Aufbau des Encoder aussieht. Jede Encoder besteht aus grundsätzlich 2 Hall-Sensoren und eines toroidale Magnetfeld. Wenn sich das toroidale Magnetfeld dreht, wird Hall-Spannung dadurch erzeugt. Wenn sich die Polarität des Magnetfeld, die Hall-Sensor erkennt, verändert, wird die Veränderung der Impulsspannung erkennt und registriert. Dadurch kann die

Winkeländrung bei drehenden Teilen erkennt werden. Mithilfe der Getriebe Übersetzung kann das Motordrehzahlen sowie die Geschwindigkeit des Roboters bestimmt werden.

2.3.3 Farbsensor

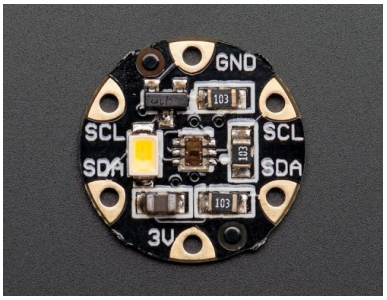


Bild 2.7 Farbsensor

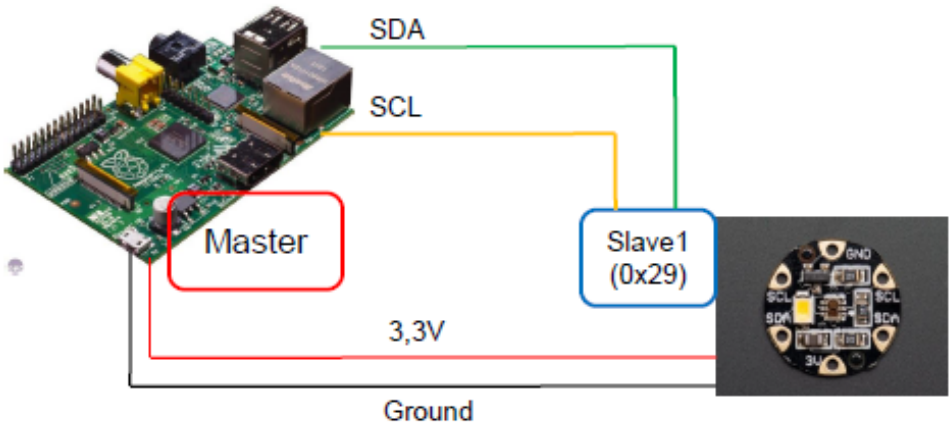


Bild 2.8 Verschaltung des Farbsensor

Das Farbsensor wird zur Farberkennung verwendet. Die Modellnummer dieses Sensors ist TAOS TCS3472. Das Farbsensor bietet eine digital Rückgabe des Rot, Grün, Blau(RGB) und klare Messwerten. Rot, Grün und Blau sind die drei Grundfarben der Farbe. Mithilfe dieser drei Farben kann jede Farbe dargestellt werden. Das Signal, das der Raspberry Pi vom Farbsensor erhält, ist das Verhältnis der drei Farben, nämlich Rot, Grün und Blau. Mithilfe des Verhältnis kann die Farbe aufgrund der RGB-Farbtabelle bestimmt werden.

Das Bild zeigt wie der Aufbau des Farbsensor TAOS TCS3472 aussieht und der Schaltplan des Farbsensor.

2.3.4 Ultraschallsensor

Bild 2.9 Ultraschallsensor

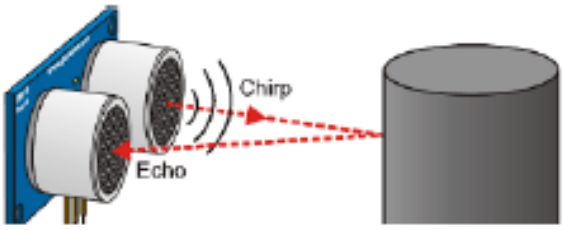


Bild 2.10 Das Verfahren der Entfernungsmessung

Das Ultraschallsensor wird zur Entfernungsmessung verwendet. Das Ultraschallsensor besteht aus hauptsächlich 2 Teile, nämlich der Sender und der Empfänger. Zuerst sendet der Sender Ultraschallsignal. Wenn ein Hindernis auftritt, wird das Ultraschallsignal zurückreflektiert, das durch der Empfänger erkannt und empfangen wird. Das Zeitintervall zwischen dem Sendesignal des Senders und dem Empfangssignal des Empfängers wird durch Raspberry Pi berechnet und dann registriert. Mithilfe des Zeitintervall und der Schallgeschwindigkeit kann die Entfernung zwischen das Hindernis und das Ultraschallsensor bestimmt werden.

Der Aufbau des Ultraschallsensor und das Verfahren, wie der Abstand zwischen das Hindernis und das Ultraschallsensor bestimmt wird, sieht wie Bild 2.10 aus.

3 Softwaretechnische Umsetzung

3.1 Softwarearchitektur

3.2 Meilenstein 1: Linie folgen

Der Hauptinhalt des Meilenstein 1 besteht darin, die Verwendung des Raspberry Pi, die grundsätzlich Verwendung der objektorientierten Programmiersprache, die Implementierung eines Qt-Projektes mit einer Benutzeroberfläche und den Funktionalitäten „Linie folgen“ und manuelle Steuerung der mobilen Plattform. Zuerst wird ein neue Qt-Projekt mit einer Benutzeroberfläche, das sogenannte GUI erstellt. Das Bild 3.1 zeigt, wie GUI aussieht. Die Benutzeroberfläche besteht aus 6 Buttons und 1 Spinbox.

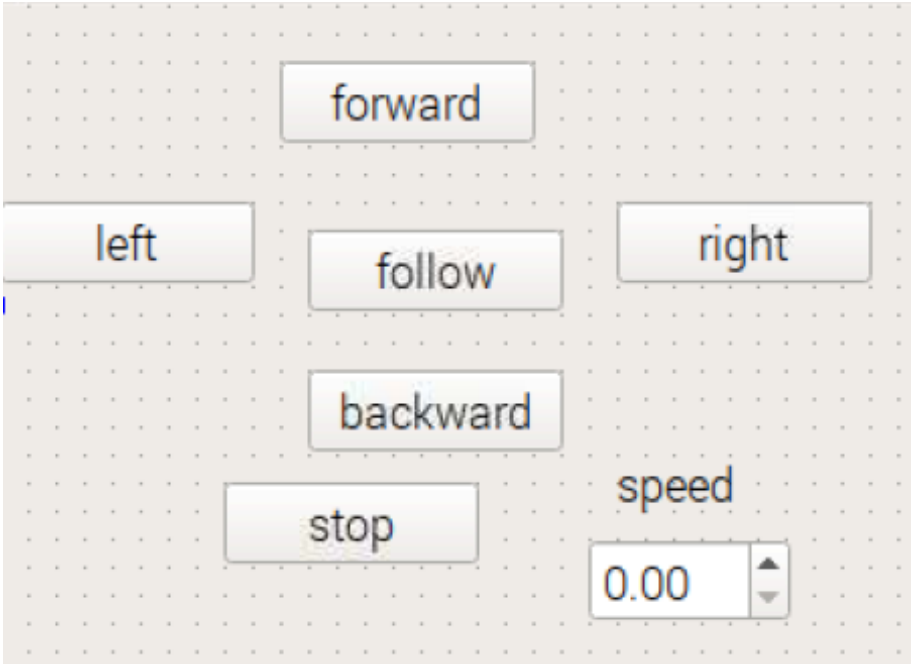


Bild 3.1 Benutzeroberfläche für Meilenstein 1

Dann wird eine Klasse “DcMotor” erstellt, um die Motoren zu kontrollieren. In dieser Klasse gibt es insgesamt 6 Methoden. Eine Methode wird verwendet, um die PIN zu initialisieren und die Verschaltung der PIN zu prüfen. Zwei Methoden wird verwendet, um die Bewegungsrichtung zu steuern. Eine Methode wird verwendet, um den Motor zu stoppen. Eine Methode wird verwendet, um die Geschwindigkeit des Motor durch die Veränderung der PWM(Puls Weiten Modulation) Wert zu kontrollieren. Eine Methode wird verwendet, um den PWM Wert des Motors zu erhalten.

Der Liniensensor wird in diesem Meilenstein verwendet. Eine Klasse “LineSensor” wird damit erstellt, um die Werte der Liniensensoren zu erhalten.

Die Klasse “MobilePlatform” ist der wichtigste Teil davon. Eine Funktion der Klasse “MobilePlatform” ist die Bewegung des Roboters manuell zu kontrollieren. Deshalb enthält diese Klasse mehrere Methoden, mit denen die Vorwärts-, Rückwärts-, Links-, Rechtsbewegung und

das Stoppen des Roboters sowie die PWM Werte der beiden Motoren gesteuert werden. Um die Linie zu folgen, eine Methode "slot$_{f}ollowline$$jwirdgeschrieben.$

Es gibt zwei Möglichkeiten, den Liniensensoren zu installieren. Eine besteht darin, den Sensor innerhalb der schwarzen Linie zu montieren, und die andere ist, den Sensor auf beiden Seiten der schwarzen Linie anzubringen.

Wenn der linke Liniensensor Weiß erkennt und der rechte Liniensensor die schwarze Linie erkennt, fährt der linke Motor vor und stoppt der rechte Motor. Wenn der linke Liniensensor die schwarze Linie erkennt und der rechte Liniensensor Weiß erkennt, fährt der rechte Motor vor und stoppt der linke Motor. Wenn die beide Liniensensoren die schwarze Linie erkennt, fahren beide Motoren vor. Wenn die beide Liniensensoren Weiß erkennt, fahren beide Motoren zurück.

Die beide Installationsmethoden werden getestet. Bei erste Methode fährt der Roboter schneller. Aber wenn der Roboter aus der Spur läuft, kann er nicht mehr zur Spur zurückkehren. Die zweite Methode löst dieses Problem. Aber diese Methode kann die Bewegung des Roboters verlangsamen.

3.3 Meilenstein 2: Sensorintegration

Der MeilenStein2 bezieht sich hauptsächlich auf die Integration des multi-Sensoren. Bei die Implantierung den mehrere Sensoren stellt das Multi-thread zu Verfügung.

Entsprechend der Aufgabezettel muss Jeder Sensor Klasse sich mit eine Thread kombinieren, weil mit dem multi-thread Algorithmus die Ressource(Speicher oder anderes) maximal benutzt werden kann. d.h. Die Geschwindigkeit der Antwort der Sensoren wird schneller als das Umstand ohne Multi-thread.

Auf dem Meilenstein 1 basiert die Programmierung des Meilenstein 2.

Insgesamt haben wir 6 Sensoren : 2XLine Sensor,1X Color Sensor, 2X Encoder, und 1 Ultrasonic Sensor. Zuerst geht es um die Befestigung aller Sensoren an der richtigen Position. Die richtige Verbindung zwischen RaspberryPi pins und Sensoren ist auch dabei erforderlich.

Bei der Programmierung wird am Anfang die GUI erweitert. Wir habe auf der GUI eine Button Gruppe für die Aktivierung der Line Editoren eingesetzt, die nach die Button Gruppe(checkbox) angelegt sind. Die Werten aller aktivierte Sensoren werden dann in dem entsprechend Linie Editoren angezeigt und aktualisiert.

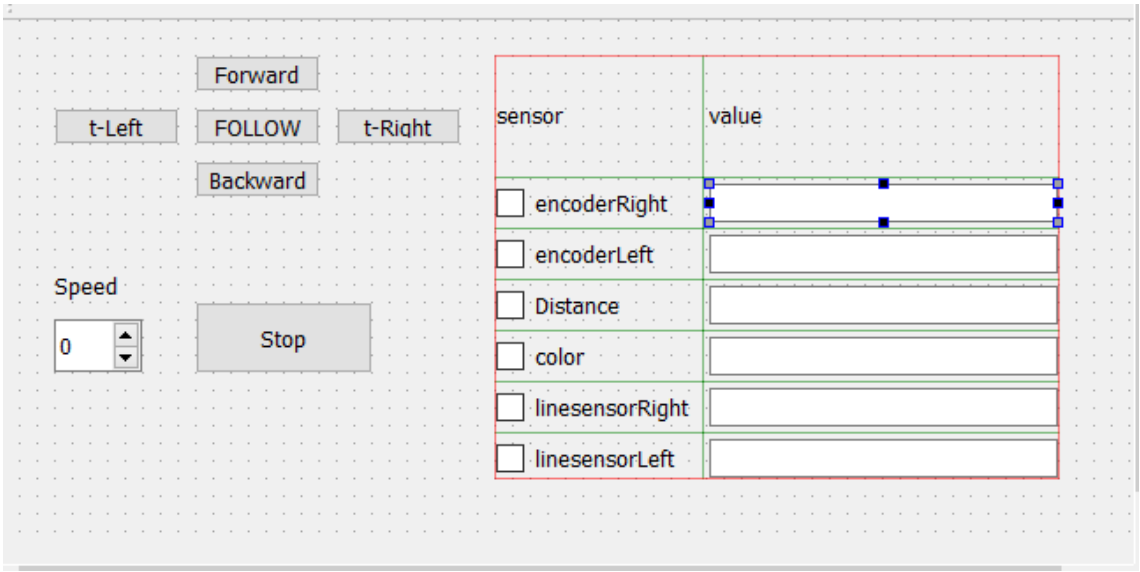


Bild 3.2 Benutzeroberfläche für Meilenstein 2

In der Klasse MainWindow sind zwei Teile Code Abschnitte eingefügt, nämlich, ein Teil für die Sendung des Signales(Anfrage erforderliches Sensorwertes), ein anderem Teil für die Erhaltung des von Objekt MobilePlatform kommenden Signals, das Parameters(Sensoren Id und Sensoren Wert) mitgenommen hat. Nach der gegebenen Sensor Id, wird entsprechender Sensor Wert auf den GUI angezeigt. Die Frequenz der Anfrage und Anzeige des Sensor Wertes ist von der einstellten Refresh Rate (durch eine QTimer) abhängig.

```
void MainWindow::createButtonGroup() {...}


void MainWindow::slot_activeLineEdit(int SensorID, bool checked)
{
    switch (SensorID) {...}
}

void MainWindow::slot_requestNewData() {...}

void MainWindow::slot_updateData(int SensorID, double sensorData) {...}
```

Bild 3.3 slots für Meilenstein 2

Dann wird eine abstrakte Klasse(AbstractSensor) auf die Projekt eingefügt. Diese Basisklasse beinhaltet Variablen, Methoden und abstrakte Methodendefinitionen, welche für alle Sensorarten relevant sind. (z.B. QTimer und aufgerufte Methode für die kontinuierliche Erhaltung des Sensor Wertes.)

Ableitung von der basischen abstrakten Klasse sind 4 Type der Sensor Klassen konkret programmiert. Die Klasse Linie Sensor ist ähnlich wie die in Meilenstein 1.In der Klasse Farbsensor sind die Kennung der Farbe mit der Umrechnung der aus 3 Farbe Kanäle kommenden Farbe Verteilung behandelt.

```
double dRedPercent = (double)nbRed/(nbRed+nbGreen+nbBlue);
double dGreenPercent = (double)nbGreen/(nbRed+nbGreen+nbBlue);
double dBluePercent = (double)nbBlue/(nbRed+nbGreen+nbBlue);
```

Bild 3.4 Farbanalyse

In der Klasse Ultrasonic Sensor und Encoder sind die Berechnung nach den Formeln wie Floge implementiert.

```
m_dSensorValue = SensortoGround+ nTimeDifference / nFactor;
```

Bild 3.5 Berechnungsformel für Ultraschallsensor

```
m_mutex_value.lock();
m_dSensorValue = (double)m_nPulseCounter/nTicsProRadumdrehung*360;
m_mutex_value.unlock();
```

Bild 3.6 Mutex

Alle Ergebnisse der Berechnung in Sensoren Klasse wird durch QMutex zu $m_dSensorValuebergeggeben.$

In der Klassse Mobileplatform haben wir 6 Instanzen der Sensoren geschafft. Jeder Instanz und ein Thread verbindet sich durch Movetothread Instruktion:

```
connectSensorThread(m_pLineSensorA, m_pThreadLineA);
connectSensorThread(m_pLineSensorB, m_pThreadLineB);
connectSensorThread(m_pColorSensor, m_pThreadColor);
connectSensorThread(m_pEncoderLeft, m_pThreadEncoderLeft);
connectSensorThread(m_pEncoderRight, m_pThreadEncoderRight);
connectSensorThread(m_pSonicSensor, m_PThreadSonic);
```

Bild 3.7 Threads

Nach der Implementierung haben wir Test gemacht, und es ergibt sich , dass alle Sensoren gut funktionieren können. Das gemisste Sensorwert ist fast gleich, verglichen mit dem Ist-Wert.

3.4 Meilenstein 3: Odometrie

Der Meilenstein 3 umfasst die Implementierung einer gesteuerten Bewegung des Roboters. Das Ziel des Meilenstein 3 ist es, den Roboter zu vermögen, auf einer geraden Linien eine definierte Wegstrecke zurücklegen und sich auf der Stelle um einen definierten Winkel zu drehen.

Die theoretische Grundlage:

In diesem Meilenstein, wird keiner Liniensensor sondern PID Regler verwendet, um die Fahrt des Roboters zu kontrollieren. Deshalb wird eine neue Klasse “Odometrie” erstellt. Das vorliegende Bild zeigt die theoretische Grundlage des PID Reglers. Zuerst wird Soll-Geschwindigkeit als Eingang eingestellt. Der Regler kann in diesem Regelkreis den eingestellten PWM Wert beeinflussen, um ein vorgegebener Wert möglichst gut eingehalten zu werden. Dazu vergleichen Regler innerhalb eines Regelkreises laufend die Größe der Soll-Geschwindigkeit mit der durch Encoder gemessenen Ist-Geschwindigkeit und ermitteln aus dem Unterschied der beiden Werte. Die sogenannte Abweichung wird als Eingang in PID Regler eingestellt.

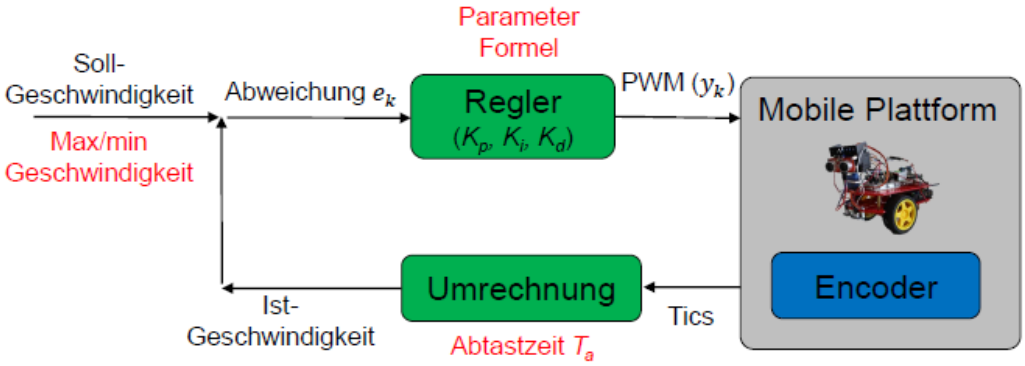


Bild 3.8 PID Regler der Geschwindigkeit

Der vollständige Name des PID Regler ist proportional–integral–derivative controller. PID Regler besteht aus den Anteilen des P-Gliedes, des I-Gliedes und des D-Gliedes. Der wichtigste Teil des PID Reglers ist die Bestimmung der Proportionalkonstante, Differenzkonstante und Integralkonstante. Der P-Anteil wird verwendet, um die Verstärkung zu reduzieren. Die Rolle des I-Anteil besteht darin, die bleibende Regelabweichung zu reduzieren. Der D-Anteil wird verwendet, um die Reaktion des Systems auf Fehler zu erhöhen, aber die Systemschwankungen erhöhen zu können. Ta ist die Abtastzeit des Geschwindigkeitsreglers.

$$y_k = Kp{\cdot}e_k + Ki{\cdot}Ta\sum_{i=0}^{k} e_i + \frac{Kd}{Ta}\left(e_k - e_{k-1}\right)$$

Bild 3.9 Berechnungsformel des PID Regler

Die tatsächliche Schritte:

1.Um die PID-Parameter besser einzustellen, wird eine neue Klasse ”Qcustomplot” erstellt. Mithilfe der neuer Klasse kann die Geschwindigkeitskurve über die Zeit gezeichnet werden.

2.Kritische-Verhältnis-Methode wird verwendet, um die PID-Parameter zu bestimmen. In dem Regelungssystem wird der Regler einen reine Proportionalregler eingestellt, und der Proportionalparameter des Reglers wird graduell von klein auf groß geändert, bis eine Oszillation mit konstanter Amplitude auftritt. Der Proportionalparameter zu dieser Zeit wird als ein kritischer Proportionalparameter bezeichnet, und das Zeitintervall zwischen zwei benachbarten Peaks wird eine kritische Oszillationsperiode Tu genannt. Stellen die Differenzkonstante und Integralkonstante auf Null ein (Ki=0, Kd=0). Erhöhen die Kp, bis eine Oszillation mit konstanter Amplitude auftritt. Zeichnen die kritische Kp und den kritischen Periodenwert Tu auf.

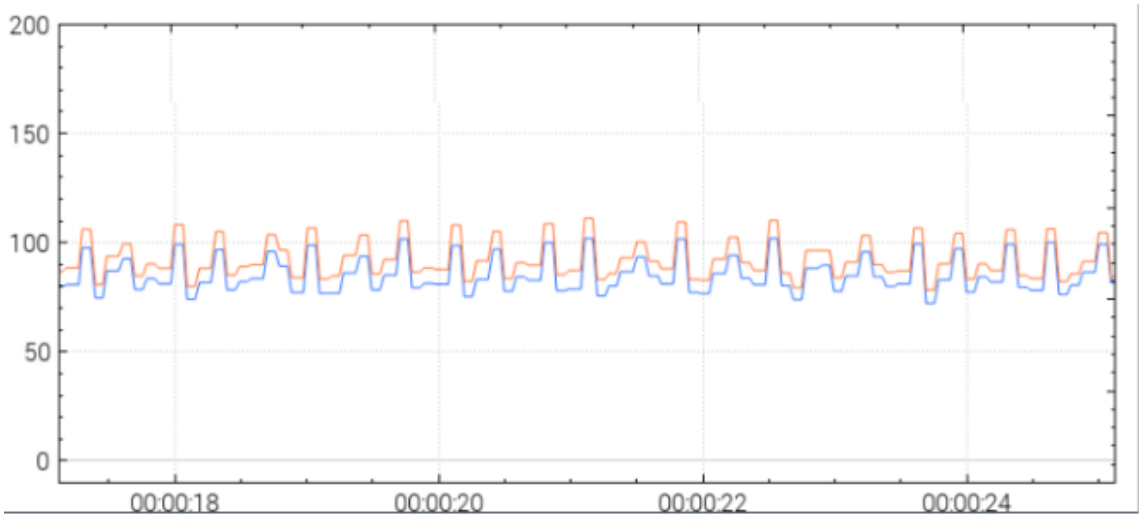


Bild 3.10 Oszillation mit konstanter Amplitude Tu=0.333s.

3.Verwenden die empirische Formel, um die Parameter des Reglers zu bestimmen.

	Kp	Kp*Ta/Ki	Kd*Ta/Kp
P	0.5*Ku		
PI	0.45*Ku	0.85*Tu	
PID	0.6*Ku	0.5*Tu	0.12*Tu

Bild 3.11 Die empirische Formel für PID Parameter

3.5 Challenge: Logistikaufgabe

Der Roboter muss automatisch entlang der Schwarzen Linien am Boden anfahren, die entsprechenden Artikel abholen und an den entsprechenden Lagerist übergeben.

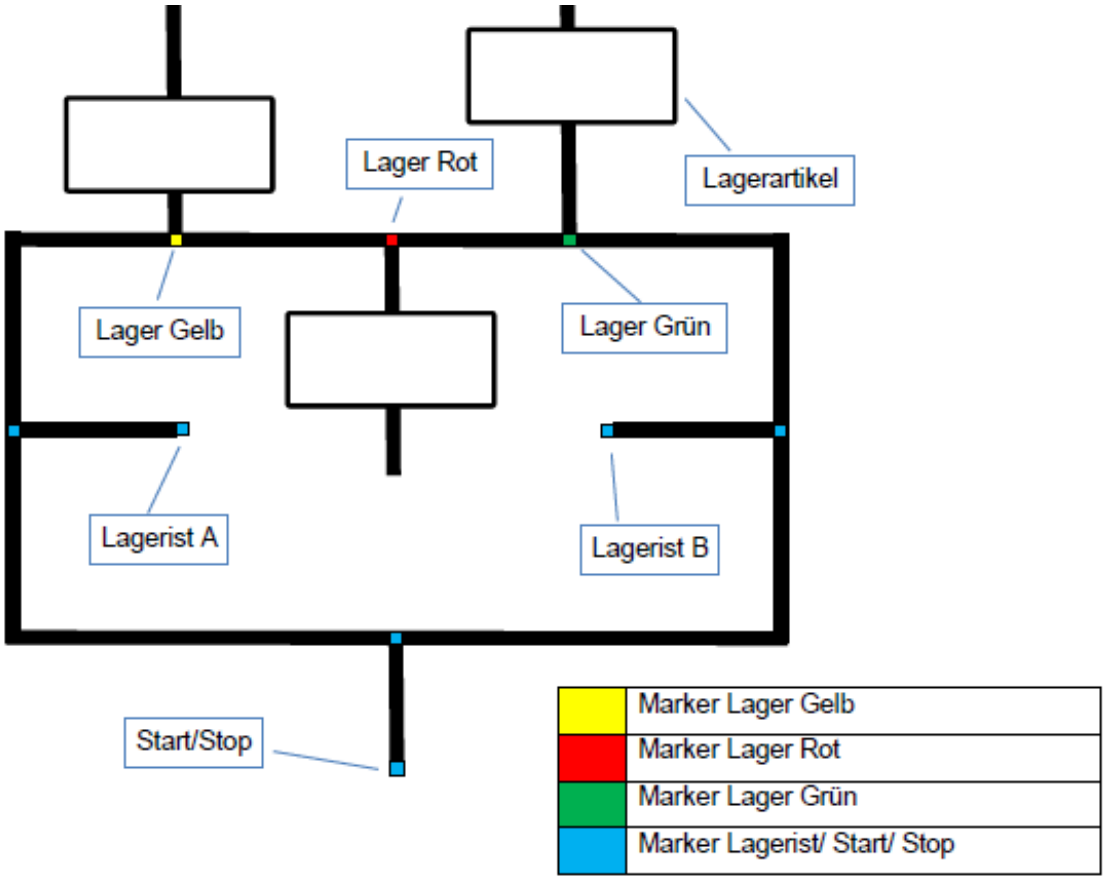


Bild 3.12

1.Der Logistikauftrag wird als XML-Datei geschrieben. Deshalb muss die XML-Datei analysiert werden. Eine neue Klasse "XML-parse" wird dazu erstellt. Und der analysierte Logistikauftrag wird als Qlist gespeichert.

2.Optimierung der Reihenfolge der Verträge

Der Logistikauftrag besteht aus hauptsächlich zwei Teile, nämlich Abholung und Lieferung. Die Abholzeit ist optimierbar und die Lieferzeit ist nicht optimierbar. Es gibt insgesamt sechs Abholformen. Nach der Reihenfolge der Zeitaufwand von kurz zu lang sind "von A nach Gelb", "von A nach Rot", "von A nach Grün" sowie "von B nach Grün", "von B nach Rot", "von B nach Gelb". Das Ziel der Optimierung ist es, möglichst viele Kombinationen "von A nach Gelb" und "von B nach Grün" zu erstellen, Kombinationen "von A nach Grün" und "von B nach Gelb" zu vermeiden. Der detaillierte Algorithmus ist unten gezeigt.

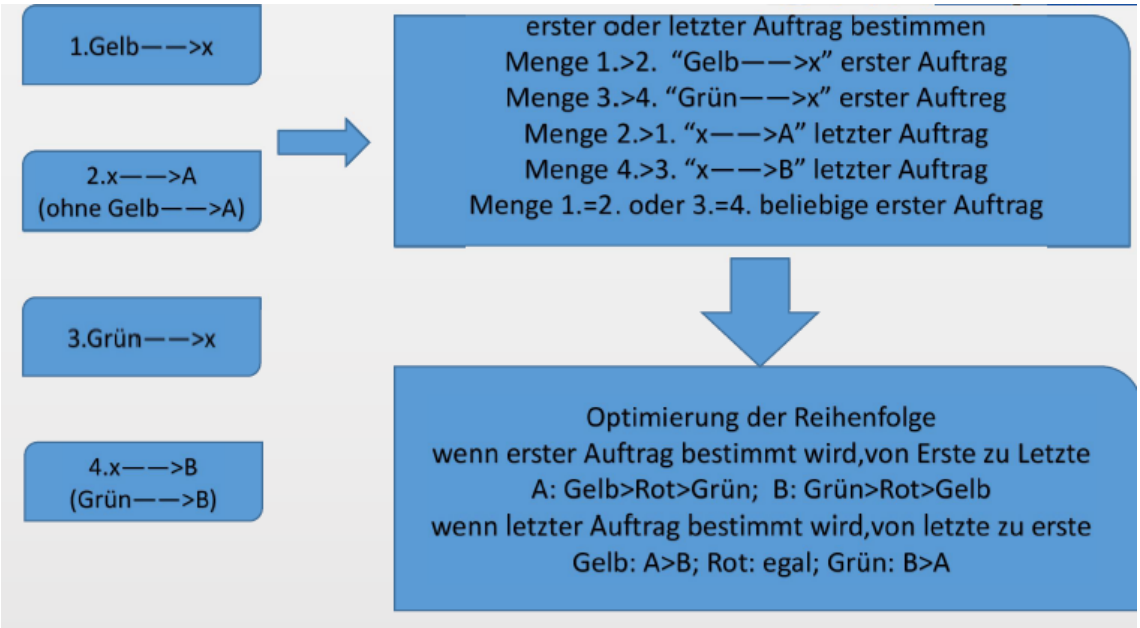


Bild 3.13

Zuerst berechnen wir die Menge des verschiedenen Auftrags. "x" bedeutet beliebige Lagerartikel oder Lagerist. Wenn die Menge "von Gelb nach x" größer als "von x nach A", wird "von Gelb nach x" den ersten Auftrag gesetzt. Wenn die Menge "von Gelb nach x" kleiner als "von x nach A", wird "von x nach A" den letzten Auftrag gesetzt. Wenn die Menge "von Grün nach x" größer als "von x nach B", wird "von Grün nach x" den ersten Auftrag gesetzt. Wenn die Menge "von Grün nach x" kleiner als "von x nach B", wird "von x nach B" den letzten Auftrag gesetzt.

Wenn erster Auftrag bestimmt werden, wird die Reihenfolge von Erste zu Letzte optimiert. Wenn der Lagerist des ersten Auftrags A ist, wird "von Gelb nach x" den zweiten Auftrag gesetzt, wenn kein "von Gelb nach x" es gibt, wird "von Rot nach x" den zweiten Auftrag gesetzt, wenn kein "von Rot nach x" es gibt, wird "von Grün nach x" den zweiten Auftrag gesetzt. Wenn der Lagerist des ersten Auftrags B ist, gilt das Gleich wie A, außer der Reihenfolge, zuerst "von Grün nach x", dann "von Rot nach x", schließlich "von Gelb nach x". Genauso wie das Bestimmen des zweiten Auftrags, werden der nächste Auftrag bis zu letzten Auftrag bestimmt.

Wenn letzter Auftrag bestimmt werden, wird die Reihenfolge von Letzte zu Erste optimiert. Wenn der Lagerartikel des letzten Auftrags Gelb ist, wird "von x nach A" den vorletzten Auftrag gesetzt, wenn kein "von x nach A" es gibt, wird "von X nach B" den vorletzten Auftrag gesetzt. Wenn der Lagerartikel des letzten Auftrags Rot ist, wird einer beliebige Auftrag den vorletzten Auftrag gesetzt. Wenn der Lagerartikel des letzten Auftrags Grün ist wird "von x nach B" den vorletzten Auftrag gesetzt, wenn kein "von x nach B" es gibt, wird "von X nach A" den vorletzten Auftrag gesetzt. Genauso wie das Bestimmen des vorletzten Auftrags, werden der nächste Auftrag bis zu ersten Auftrag bestimmt.

3.Zustandsautomaten Wir haben dann eine Zustandsautomaten in der Klasse Mobileplatform eingefügt. Es gibt gesamt 5 Zustände in eine Zustand Gruppe und eine zusätzliche Final Zustand. Wenn die Bedienung in der Zustand 4 erfüllt(restliche Aufträge Nummer ist gleich 0), wird eine Signal gesendet und springt der Zustandsautomaten aus die Zustand Gruppe.

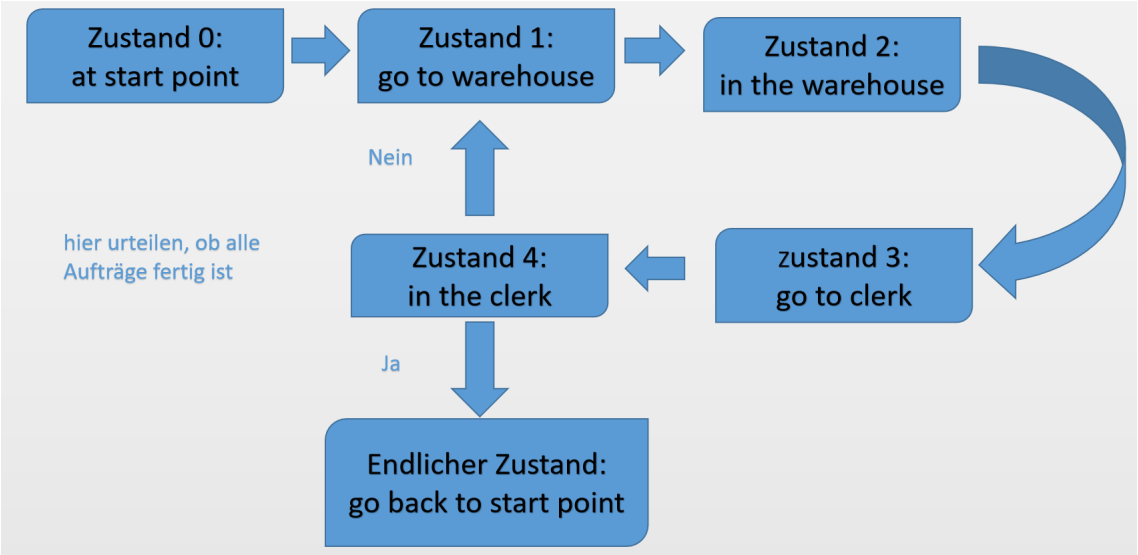


Bild 3.14 Zustandsautomat

Die gewünschte Aufgabe zu ausführen, benötigen wir bei jedem Zustand ein slot-todo()(in der Klasse Mobileplatform).

```
void MobilePlatform::slot_GotoLager() {...}

void MobilePlatform::slot_getWare() {...}

void MobilePlatform::slot_GotoClerk() {...}

void MobilePlatform::slot_putdownWare() {...}

void MobilePlatform::slot_GobackStartpoint() {...}
```

Bild 3.15 Teil des Programms über Slots

Bei jedem Slot gibt es einige verschiedene Cases und alle Cases unter slot sind wie folgend ausgelistet:

Slot-gotoLager():

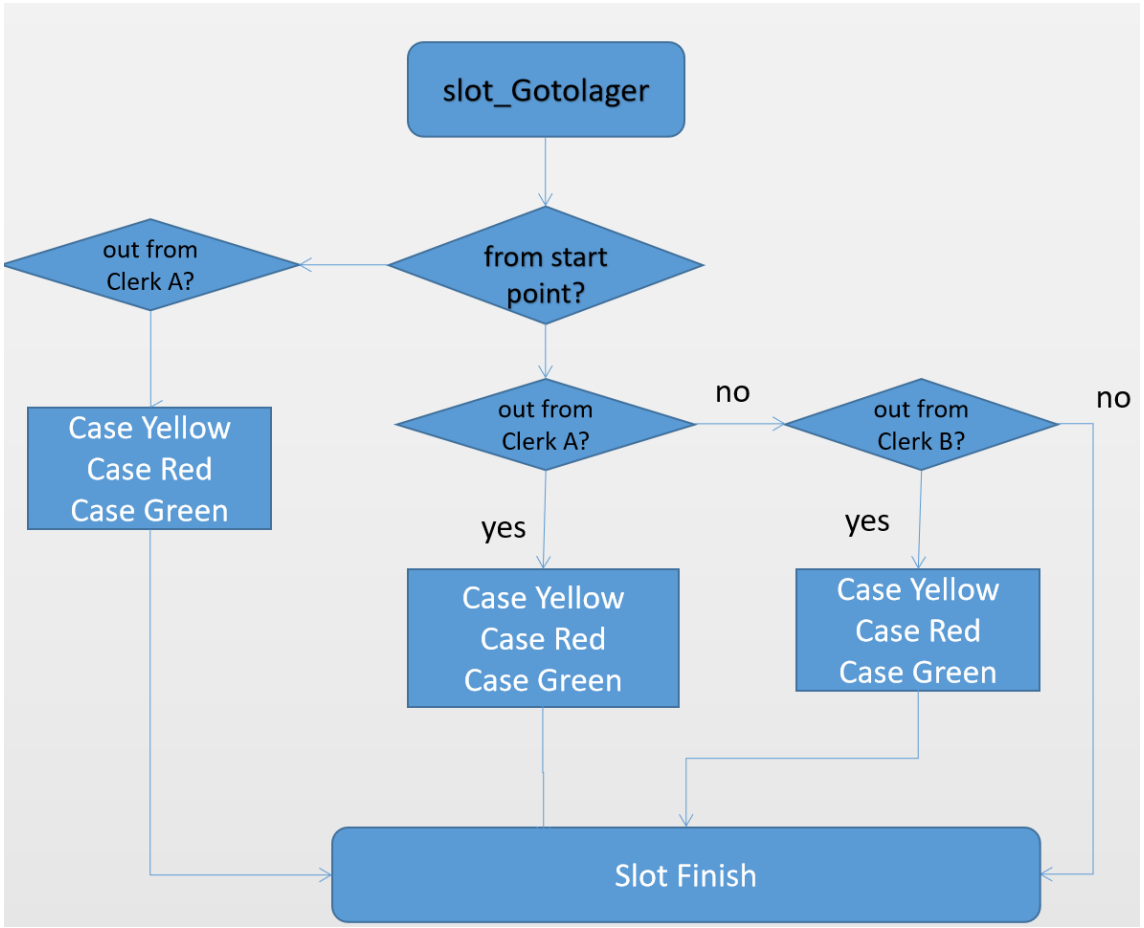


Bild 3.16 Flussdiagramm des Slot-Gotolager

Slot-GotoClerk():

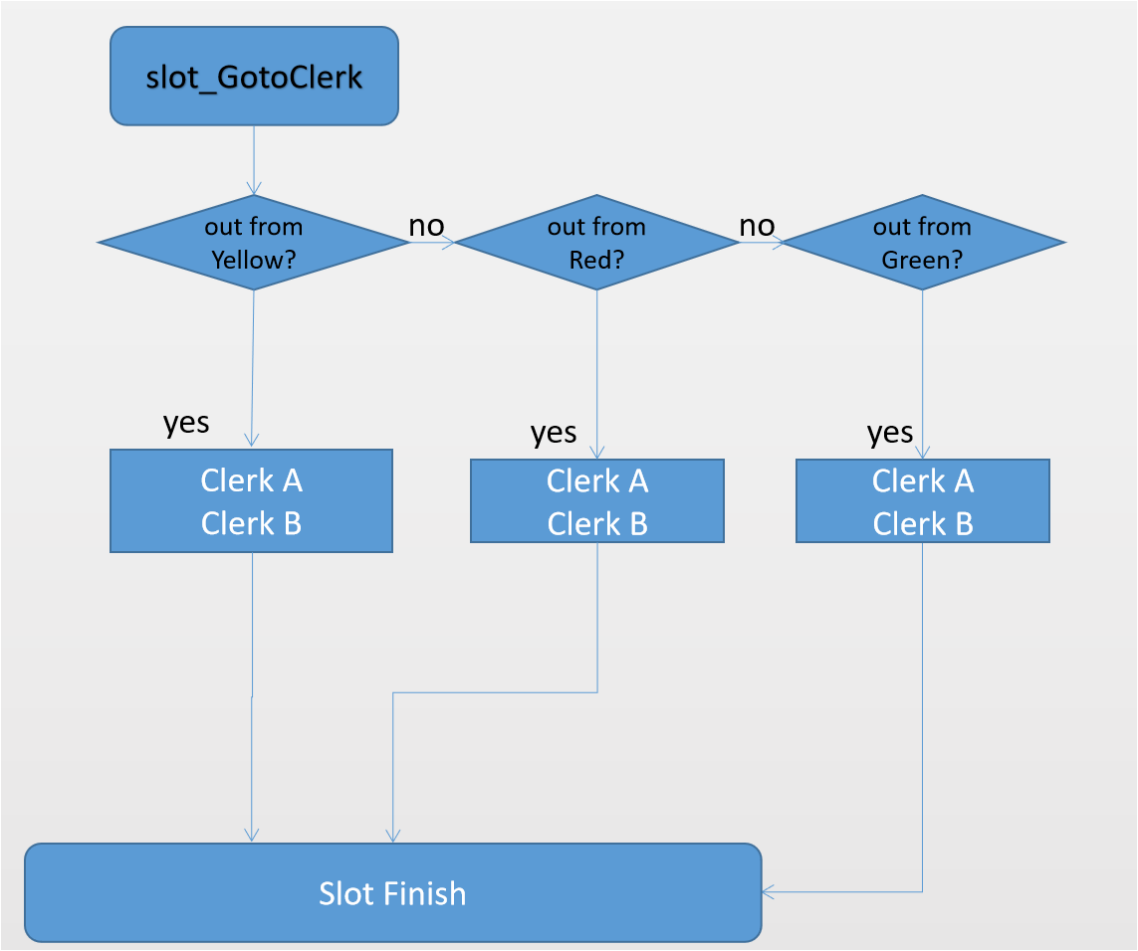


Bild 3.17 Flussdiagramm des Slot-GotoClerk

Nach dem parsen des XML-Datei erfahren wir den momentan ausgeführten Auftrag und der aktuelle Umstand.

Slot-getWare(): gostraight und stop while (Hight<50), turn around,back straight und stop while(Colorchanged).

Slot-putdownWare(): ähnlich wie obere slot.

In jedem Case gibt es 2 Arten der Bewegungen, nämlich wie folgend, ControlledStraight und ControlledTurn. Und in Controlled Straight existiert auch 2 Arten von Linie Bewegung: PID controlled (Wenn wir muss die Länge der Bewegung kontrollieren) oder Linie Folgen(bei den restlichen Umständen). Die gesamte Bewegung besteht aus diese Verschiedene Arten der Bewegungen.

```
void MobilePlatform::ControlledStraight(double expectedvelocity, double destination, double targetColor,\
                                        double verticaldistance,bool openline) {...}

void MobilePlatform::ControlledTurn(double rotationvelocity, double angle) {...}
```

Bild 3.18 Methoden für 2 Arten der Bewegungen des Roboters

Am Anfang der Implementierung haben wir die Nummer die gesamte Aufträge gelesen und habe dafür eine Counter eingesetzt. Wenn Counter gleich 0 ist, wird die Zustandsautomat zum Finale Zustand gehen und machen das Slot-GobackStartpoint.

4 Diskussion

1.Anbringung der Liniensensoren Es gibt zwei Möglichkeiten, den Liniensensoren anzubringen. Eine besteht darin, den Sensor innerhalb der schwarzen Linie anzubringen, und die andere ist, den Sensor auf beiden Seiten der schwarzen Linie anzubringen. Wie das Bild zeigt, sind die beiden Installationsmethoden.

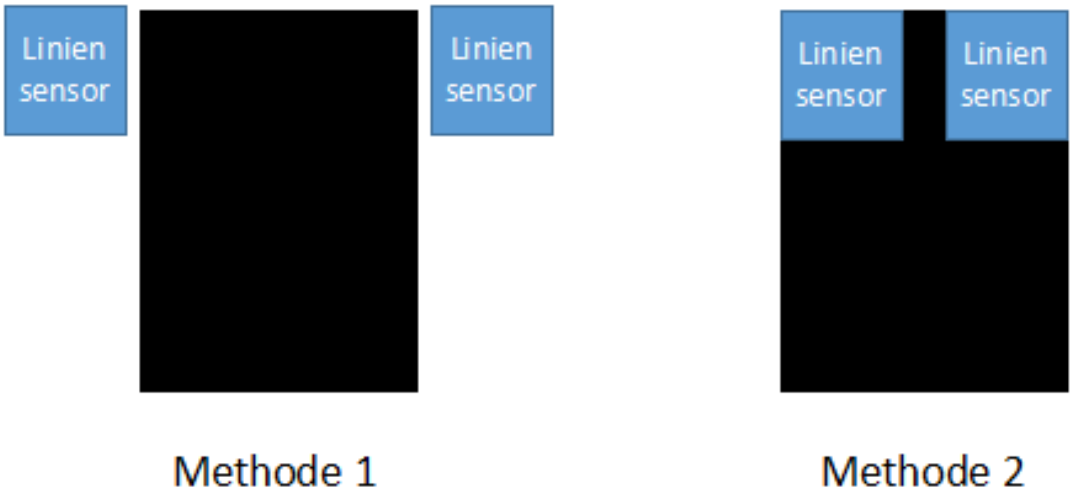


Bild 4.1 2 Möglichkeiten für Anbringung der Liniensensoren

Der Roboter fährt schneller bei der Methode 1. Aber die Route des Roboters ist mehr gerade bei der Methode 2. Bei der Methode 2 kann der Roboter nicht um 90 Grad drehen. Am Ende haben wir für die Methode 1 entschieden.

2.Mischung aus PID-Regler und Liniensensoren

Bei Challenge haben wir PID-Regler mit Liniensensoren zu kombinieren versucht. Aber die Route, die der Roboter fährt, ist besonders unordentlich. Die gemessene Geschwindigkeit des Roboters ist sehr schwankend. Der Grund liegt darin, dass es gegenseitige Störungen zwischen PID-Regler und Liniensensoren gibt.

3.Die Verwendung der invokeMethod

In der Klasse “MobilePlatform” werden einige von der Klasse “Odometrie” vererbten Methoden aufgerufen, um den QTimer in Klasse “Odometrie” zu erstellen und dann schließen. Aber es gibt immer Warning: “QTimer cannot be closed from another thread.” Das heißt, die gegenseitige Interferenz und Verwirrung zwischen verschiedenen Threads werden verursacht. Um diese Problem zu lösen, werden die invokeMethod verwendet. Während des Aufrufs durch die invokeMethod werden das Programm in gleiche Thread bleiben, um die Verwirrung zwischen der Threads zu vermeiden.

4.Die Analyse der Gründe für das Scheitern

Bei Challenge sind wir gescheitert.

1)Zuerst wird die angegebene XML-Datei falsch analysiert. Der Grund ist, wir haben vergessen, “std::cout” zu löschen. In der Beispieldatei gibt es 3 Teilaufträge. Deshalb werden 3 “cout” geschrieben, um den Auftrag zu zeigen. Aber die angegebene XML-Datei besteht aus 2 Teilaufträge. Das heißt, es gibt eine “cout”, die nichts ausgeben kann. Wenn das Path der angegebenen XML-Datei importiert wird und das Programm läuft, wird das Programm “crashed”.

2)Die Zustandsautomaten werden direkt in Klasse “MobilePlatform” geschrieben. Es gibt Möglichkeit, dass Threads blockiert werden. Die Verbesserungsmethode ist, eine neue Klasse “Statesmaschine” zu erstellen, noch ein neue Thread für Zustandsautomaten zu erstellen.

3)Das Farbsensor reagiert sehr langsam.

5 Zusammenfassung

In der Lehrveranstaltung haben wir viel Kenntnisse über den Raspberry Pi gelernt. Darüber hinaus werden verschiedene Sensoren vorgestellt, nämlich Liniensensor, Farbsensor, Encoder und Ultraschallsensor. Durch die tatsächliche Benutzung haben wir das Arbeitsprinzip der Sensoren und die Verwendungsmethoden beherrscht. Am wichtigsten ist, dass wir in dieser Lehrveranstaltung die Grundkenntnisse und verschiedene Programmiermethoden der C++ Programmiersprache beherrscht haben.

Als eine objektorientierte Programmiersprache zeigt C++ die Vorteile in dem Programm. Wenn man neue Funktionen implementieren will, wird einfach neue Klasse erstellt, anstatt ganzen Programm neu zu schreiben. Es kann nicht nur Zeit sparen, sondern auch viele Probleme vermeiden.

Die Dozentin und die Betreuer haben uns ausreichende sinnvolle Hilfe zur Verfügung gestellt. Zum Beispiel hat Vincent uns mit der Verwendung der invokeMethod erklärt, um die Verwirrung der Threads zu vermeiden. Es funktioniert sehr gut.

Ein weiterer guter Punkt ist die Teamarbeit, die die Effizienz deutlich verbessern kann. Darüber hinaus kann man auch viel von seinen Kollegen lernen.

Literatur

[1] Helmut Balzert: *Raspberry Pi: Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Springer Verlag, 2009.

[2] Upton, Eben, and Gareth Halfacree: *Raspberry Pi: Einstieg und User Guide*. MITP-Verlags GmbH & Co. KG, 2013.

[3] Bjarne Stroustrup: *Einführung in die Programmierung mit C++*. Pearson Studium, Addison-Wesley, 2010.

[4] https://de.wikipedia.org/wiki/Raspberry$_Pi, 2018. $*https : //de.wikipedia.org/wiki/Hall - Effekt*, 2018.