Name: _____ Klasse: ___ Datum: _____

Aufgabe 1

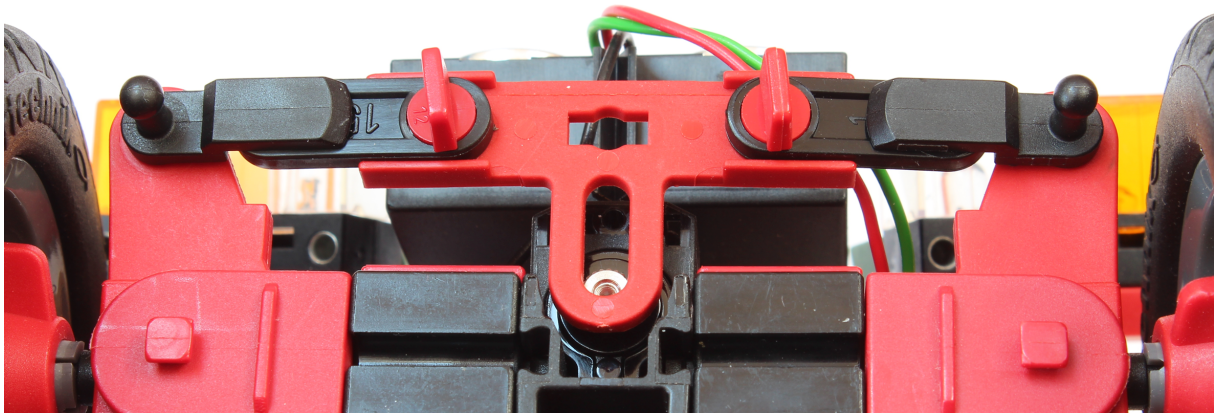
Tachometer, Hodometer und Taxameter

Das Fahrzeug wird aufgebaut und mit einem Tachometer (Geschwindigkeitsanzeige) und einem Hodometer (Kilometerzähler) ausgestattet. Ergänzend kannst du es mit einem Taxameter versehen.

Konstruktionsaufgabe

Baue das Fahrzeug nach der Bauanleitung auf. Schließe zunächst nur den Motor (M1) und den Encoder (C1 und 9V-Spannungsausgang) an den TXT an (siehe Schaltplan). In dieser Aufgabe wird das Fahrzeug mit einem Tachometer (zur Anzeige der Geschwindigkeit), einem Hodometer (Kilometerzähler) und einem Taxameter (Fahrtkostenberechnung) ausgestattet.

Beachte: Den Servo-Hebel solltest du zunächst noch nicht befestigen; die nächste Aufgabe beschäftigt sich mit der Geradeausfahrt. Einfacher ist es, die rote Servo-Lasche falsch herum zu montieren, um die Lenkung zu arretieren:



Montage Servo-Lasche.jpg

Tipp: Zum Testen deiner Programme empfiehlt es sich, das Fahrzeug zunächst „aufzubocken“, d. h. die Unterseite des Fahrzeugs oder die hintere Stoßstange so zu unterlegen, dass die Hinterräder nicht den Boden berühren.

Programmieraufgaben

1. Anzeige der Encoder-Werte

Schreibe zunächst ein Programm, das den Motor startet und die seit dem Start gezählten Impulse des Encoders (C1) anschließend kontinuierlich auf dem Display des TXT anzeigt.

2. Hodometer

Aus den Impulsen des Encoders sollst du nun die vom Fahrzeug zurückgelegte Strecke in Metern berechnen und auf dem Display des TXT anzeigen (Hodometer oder Kilometerzähler).

2a. Dazu musst du zunächst die Übersetzung des Getriebes bestimmen und den Umfang der Bereifung messen (siehe auch Aufgabe 6 des Robotics TXT 4.0 Base Set). Auf welche Werte kommst du? Gib eine Formel zur Umrechnung der Encoder-Impulse in die zurückgelegte Strecke (in Metern) an.

2b. Schreibe ein einfaches Testprogramm, mit dem du die Genauigkeit der Messung entlang einer bspw. 2 m langen Teststrecke überprüfen kannst. Korrigiere erforderlichenfalls den Umrechnungsfaktor.

2c. Passe dein Blockly-Programm aus Programmieraufgabe 1 und die Display-Ausgabe so an, dass die zurückgelegte Strecke angezeigt wird.

2d. Der Zählereingang kann einen maximalen Wert von 65.535 annehmen. Wie lang ist demnach die von deinem Kilometerzähler maximal messbare Fahrstrecke (in Metern)?

3. Tachometer

Neben der zurückgelegten Distanz soll nun auch die Momentangeschwindigkeit des Fahrzeugs auf dem Display des TXT angezeigt werden (Tachometer).

3a. Zur Bestimmung der Geschwindigkeit musst du die in einer festen Zeiteinheit (z. B. in einer Sekunde) zurückgelegte Strecke bestimmen. Gib eine Formel zur Berechnung der Geschwindigkeit an.

3b. Erweitere dein Blockly-Programm aus Programmierausgabe 2 um die Berechnung der Geschwindigkeit und zeige auf dem Display des TXT neben der zurückgelegten Strecke auch die aktuelle Geschwindigkeit an.

Experimentieraufgaben

1. Tachometer und Hodometer nebenläufig

Die Anzeige von zurückgelegter Strecke und Geschwindigkeit soll nun in einem nebenläufigen Programm (*Thread*) parallel zum eigentlichen Steuerprogramm des Fahrzeugs erfolgen.

Schreibe das Programm so um, dass die Anzeige parallel zum Hauptprogramm erfolgt und kontinuierlich aktualisiert wird. Der Motor soll im Hauptprogramm gestartet werden.

2. Taxameter

Damit das autonome Fahrzeug später auch als Taxi genutzt werden kann, erhält es nun ein Taxameter. Montiere einen Taster an deinem Fahrzeug und schließe ihn an I8 an.

Wenn der Taster zum ersten Mal gedrückt wird, soll das Taxameter starten und den Fahrpreis auf dem Display des TXT anzeigen. Wird der Taster zum zweiten Mal gedrückt, soll das Taxameter angehalten und der zu zahlende Preis angezeigt werden

2a. Der Preis setzt sich aus einer Grundgebühr und einem streckenabhängigen Tarif zusammen. Lege Grundgebühr und Streckentarif in einer Variablen fest.

2b. Zeichne ein Zustandsübergangsdiagramm für das Taxameter.

2c. Erweitere dein Blockly-Programm entsprechend. Auch die Grundgebühr und der Streckentarif sollen auf dem Display angezeigt werden.

Anlagen

Aufgabe 1: Tachometer, Hodometer und Taxameter

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Maßband oder Zollstock, Papierstreifen (für die Messung des Reifenumfangs).

Weiterführende Informationen

[1] Andreas Wolf: *Tachometer. Die Geschichte eines unverzichtbaren Instruments*. 04.08.2014.

[2] Alper Aribal (SeoRocket): *Taxameter*. DeWiki.de.

Aufgabe 1: Tachometer, Hodometer und Taxameter

In dieser Aufgabe wird zunächst ein Fahrzeug mit Hinterradantrieb und Achsschenkellenkung (Servo-Motor) konstruiert. Zur Vorbereitung auf die autonome Steuerung werden ein Tachometer (mit Geschwindigkeitsanzeige) und ein Hodometer (Anzeige der gefahrenen Strecke) programmiert. Schließlich kann das Hodometer zu einem Taxameter ausgebaut werden.

Thema

Fahrzeugkonstruktion, Auswertung des Encoders zur Geschwindigkeitsbestimmung, Fahrpreisberechnung und nebenläufige Prozesse (Threads).

Lernziele

- Fahrzeugkonstruktion: Achsschenkellenkung, Differentialantrieb
- Umrechnung der Encoderimpulse in Distanzen (Übersetzung, Radumfang), Fahrpreisberechnung
- Umrechnung der Encoderimpulse in Geschwindigkeiten
- Nebenläufige Prozesse (Threads)

Zeitaufwand

Das autonome Fahrzeug wird nach Bauanleitung aufgebaut. Je nach Erfahrung der Schülerinnen und Schüler mit fischertechnik werden dafür ein bis zwei Unterrichtsstunden (45-90 Minuten) benötigt.

Für die Entwicklung der Programme zur Lösung der Programmieraufgaben benötigen Schülerinnen und Schüler ein bis zwei Unterrichtsstunden (45-90 Minuten). Ggf. sollten Hilfestellungen zur Umrechnung der Encoder-Impulse gegeben werden.

Erfahrene Schülerinnen und Schüler können in der Experimentieraufgabe ein Taxameter ergänzen. Die Lösung dieser Experimentieraufgabe ist für das Verständnis der weiteren Aufgaben nicht erforderlich.

Bezug Curriculum

Land	Stufe/Fächer	Bezüge
BW	SEK I	GYM 8/9/10 NWT-3.2.4.3 Steuerungsabläufe (Ampelsteuerung, Robotik) (7), Informationsverarbeitung - Autonomes Fahren (8), S.27; IMP 8-3.1.1.2 Algorithmen (1), S. 28ff; INFWF 8-3.1.2 Algorithmen (1), S. 15; INFWF 9-3.2.2 Algorithmen (2), S. 21; INFWF 10-3.3.2 Algorithmen (2), S. 28;
BY	SEK I	RS- IT 2.7 Logik und Robotik, S.699; GYM 9/10 LPLUS INF - Modellieren, Implementieren, Anwenden, Softwareprojekte
BE	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
BB	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
HB	SEK II	GYM OS INF-Algorithmen und Datenstrukturen, S. 6; GYM OS INF-Imperative Programmierung, S. 7
HH	SEK I	Stadtteil 9/10 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 20; GYM 9 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 19
HE	SEK I	ohne curricularen Vorgaben
MV	SEK I	GYM 5 INF-3 Programmieren? Kinderleicht!, S.16; GYM 6 INF-3 Entscheidungen treffen und Spiele gestalten, S.19 GYM 7 INF-3 Spiele entwickeln, S.22; GYM 8 INF-3 Sensorgesteuerte Anwendungen entwickeln, S.25; GYM 9 INF-3 Problemlösen durch Programmieren, S.32
NI	SEK I	KC-INF LF Algorithmisches Problemlösen; S.19; KC-INF LF Automatisierte Prozesse, S.22; SEK 2 KC-INF LF1 Algorithmen und Datenstrukturen, S.14; SEK 2 KC-INF LF1 Informationen und Daten, S.16; ; SEK 2 KC-INF LF1 Automaten und Sprachen, S.19
NW	SEK I, II	RS 9/10 WPF TECHNIK 2.3 Inhaltsfeld 7: Kommunikations- und Digitaltechnik S.23; 5/6 KLP INF - Algorithmen, S. 17, 18; 5/6 KLP INF - Automaten und künstliche Intelligenz, S. 18; SEK 2 KLP GOS INF - 2 Algorithmen, S. 21 ff; KLP GOS INF - 3 Formale Sprachen und Automaten, S. 22
RP	SEK I	IPS 5 INF - Informatiksysteme und Netze, S. 7; IGS/GYM INF-2.1 Grundlagen der Informationsverarbeitung, S. 17; IGS/GYM INF-2.2 Algorithmisches Problemlösen, S. 20
SL	SEK I, II	GYM 9 INF - Imperative Programmierung, S. 3; INF - Algorithmik, S. 3; GYM OS INF GOS-Funktionsweise von Computersystemen, S.9ff.

SN	SEK I	GYM 7 INF LB 3: Computer verwenden – Komplexaufgabe, S. 7; GYM 8 INF LB 2: Daten verarbeiten, S.10
ST	SEK I, II	GYM 9 INF 3.2 Algorithmen interpretieren und entwickeln, S.15 ff.; GYM 11/12 INF 3.4 Kurs 3 Software Engineering und Projektarbeit, S. 23
SH	SEK I	INF PB1 Modellieren und Strukturieren, S. 12; INF PB2 Implementieren, Programmieren, Realisieren, S. 13; ; FA Physik, Variabilität S.13
TH	SEK I	GYM 10 INF - 2.3 Algorithmen, S. 14 ff.; GYM 10 INF 2.5.1 Technische Informatik, S. 18ff.

Anlagen
Aufgabe 1: Tachometer, Hodometer und Taxameter

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Maßband oder Zollstock, Papierstreifen (für die Messung des Reifenumfangs).

Weiterführende Informationen

[1] Andreas Wolf: *Tachometer. Die Geschichte eines unverzichtbaren Instruments*. 04.08.2014.

[2] Alper Aribal (SeoRocket): *Taxameter*. DeWiki.de.

Name: _____ Klasse: ___ Datum: _____

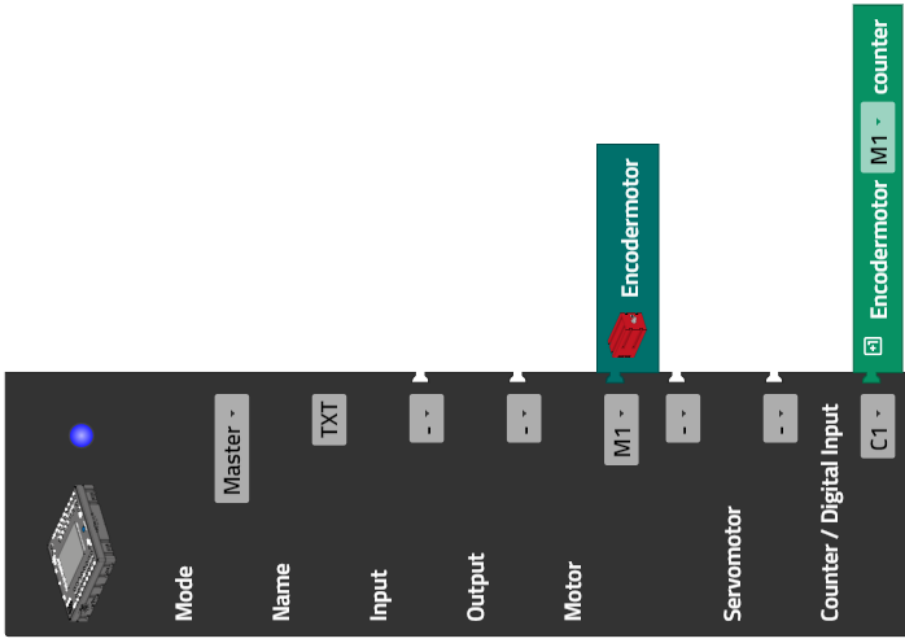
Lösungsblatt Aufgabe 1

Tachometer, Hodometer und Taxameter

Beim Tacho- und beim Hodometer muss zunächst aus Reifenumfang, Encoder-Signalen und der Übersetzung des Getriebes die je Impuls zurückgelegte Entfernung bestimmt werden. Keine schwierige Rechnung, aber eine Übung in der Umrechnung von Einheiten. Die Geschwindigkeitsangaben erfolgen bei den folgenden Lösungsvorschlägen in m/h; sie können aber auch in km/h umgerechnet werden.

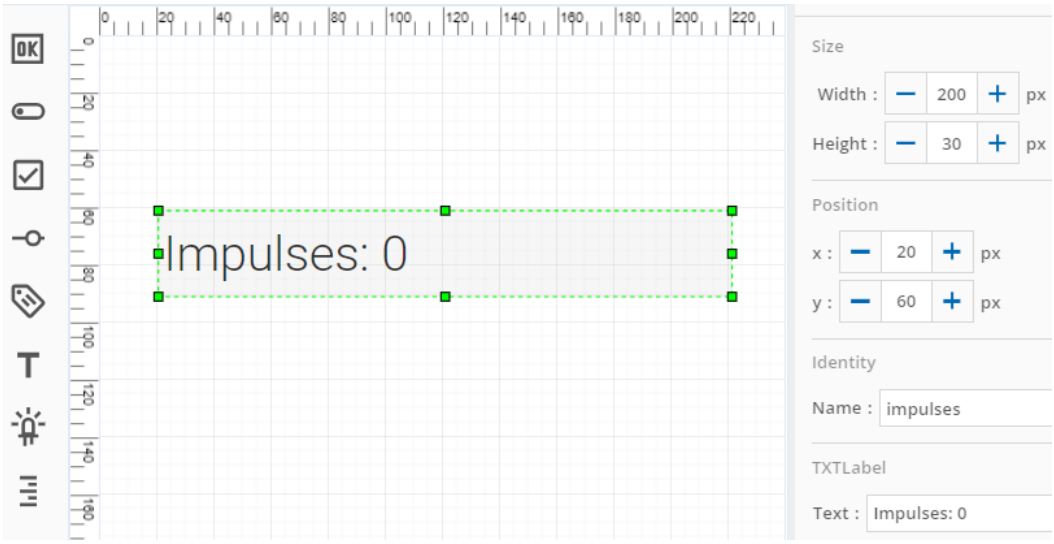
Programmieraufgaben

Konfiguration der Aktoren (in dieser Aufgabe wird nur der Encodermotor benötigt):

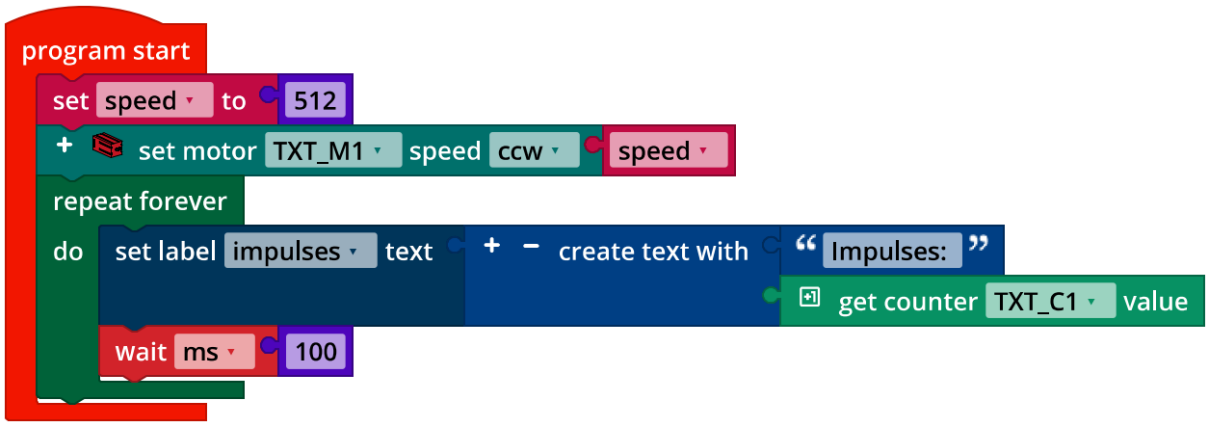


1. Anzeige der Encoder-Werte

Display-Konfiguration:



Programm (Beispiel) zur Anzeige der Zählerwerte des Encoders:



Display_Counter_Encoder.ft

Die Pause in der Schleife ist sinnvoll für eine kontrolliert-kontinuierliche Anzeige auf dem Display. Lässt man sie weg, versucht der Controller die Verzögerung durch die Anzeige auf dem Display auszugleichen.

2. Hodometer

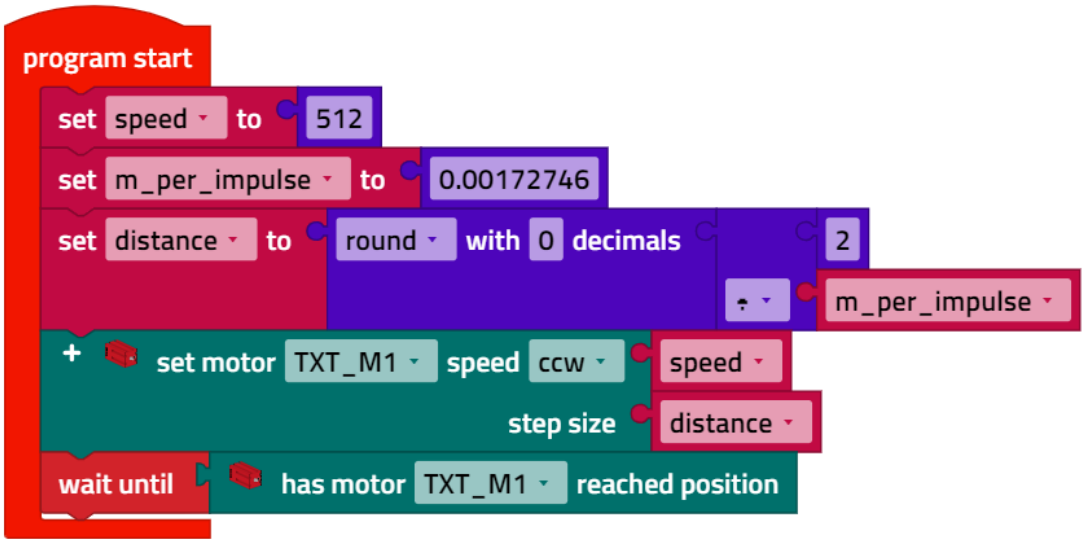
Der Magnetencoder des Motors liefert 63,9 Impulse je Umdrehung der Abtriebsachse. Das Getriebe des Differenzialkäfigs übersetzt 14:26 = 7:13 ins Langsame.

2a. Der Umfang des Reifens liegt bei ca. 20,5 cm (siehe Aufgabe 6 des Robotics TXT 4.0 Base Set; empfehlenswerte Messmethode: Papierkante um den Reifen legen, Umfang markieren und dann Länge der Papierkante bis zur Markierung messen).

Aus den über eine Fahrstrecke gezählten Impulsen i lässt sich die Distanz d (in m) also nach folgender Formel bestimmen:

$$d = i \cdot \frac{1}{63{,}9} \cdot \frac{7}{13} \cdot 20{,}5 \cdot \frac{1}{100}\ \text{m} \approx i \cdot\ 0{,}00172746\ \text{m}$$

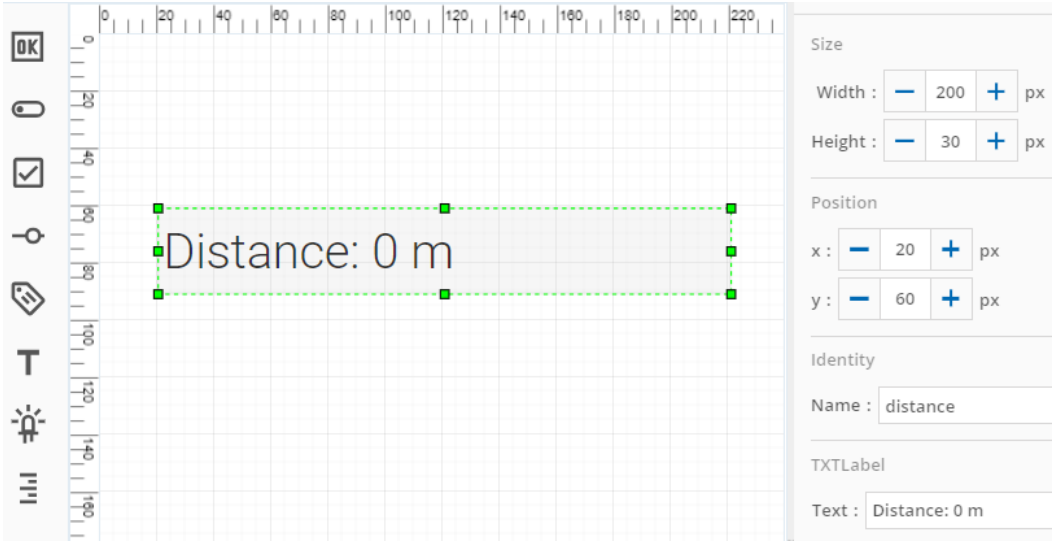
2b. Einfaches Testprogramm zur Überprüfung der Messung:



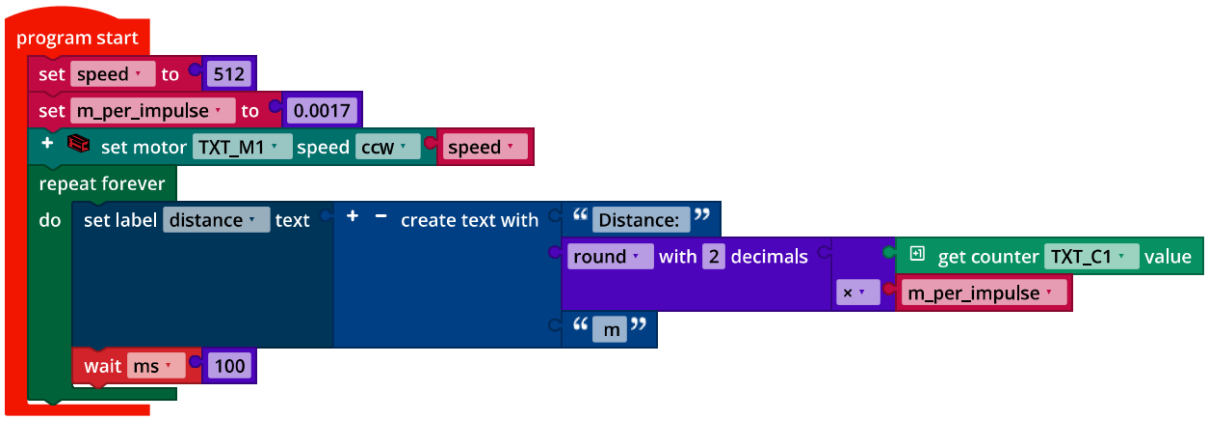
Distance_Test.ft

In Tests stoppt das Fahrzeug ein Stückchen vor Erreichen der 2-m-Marke. Der Umrechnungsfaktor wurde daher auf 0,0017 korrigiert.

2c. Display-Konfiguration:



Programm (Beispiel) Hodometer:



Odometer.ft

2d. Die maximal messbare Fahrstrecke d_{max} liegt bei

$$d_{max} = 65.535 \cdot 0{,}0017 \text{ m} \approx 111{,}41 \text{ m}$$

3. Tachometer

Als festes Zeitintervall zur Bestimmung der Geschwindigkeit bietet sich z. B. 1 s an: lang genug für eine genaue Messung und hinreichend kurz, um auf Geschwindigkeitsänderungen schnell zu reagieren.

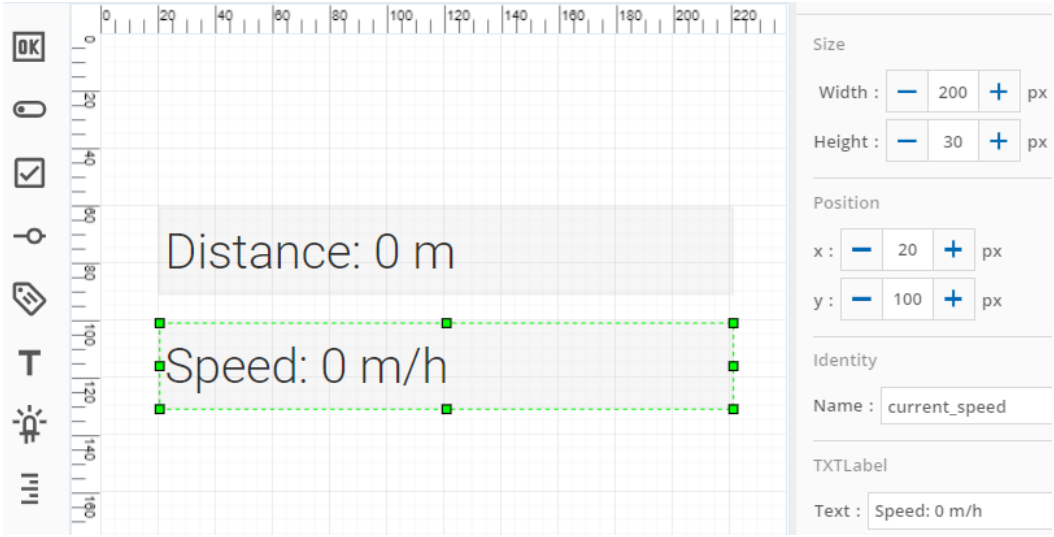
3a. Damit lässt sich die Geschwindigkeit v aus der während einer Sekunde gezählten Impulse i wie folgt berechnen:

$$v = i \cdot \frac{1}{63{,}9} \cdot \frac{7}{13} \cdot 20{,}5 \cdot 36 \ {}^{\mathrm{m}}/_{\mathrm{h}} \approx i \cdot 6{,}2189 \ {}^{\mathrm{m}}/_{\mathrm{h}}$$

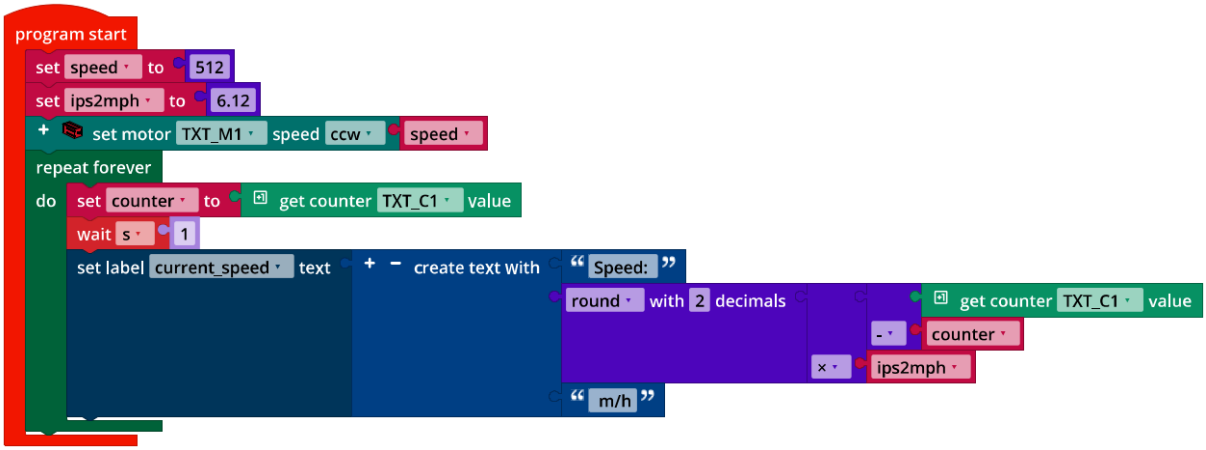
Wenn wir den korrigierten Wert aus Programmieraufgabe 2 berücksichtigen, ergibt sich:

$$v = i \cdot 0{,}0017 \cdot 3600 \ {}^{\mathrm{m}}/_{\mathrm{h}} \approx i \cdot 6{,}12 \ {}^{\mathrm{m}}/_{\mathrm{h}}$$

3b. Display-Konfiguration:

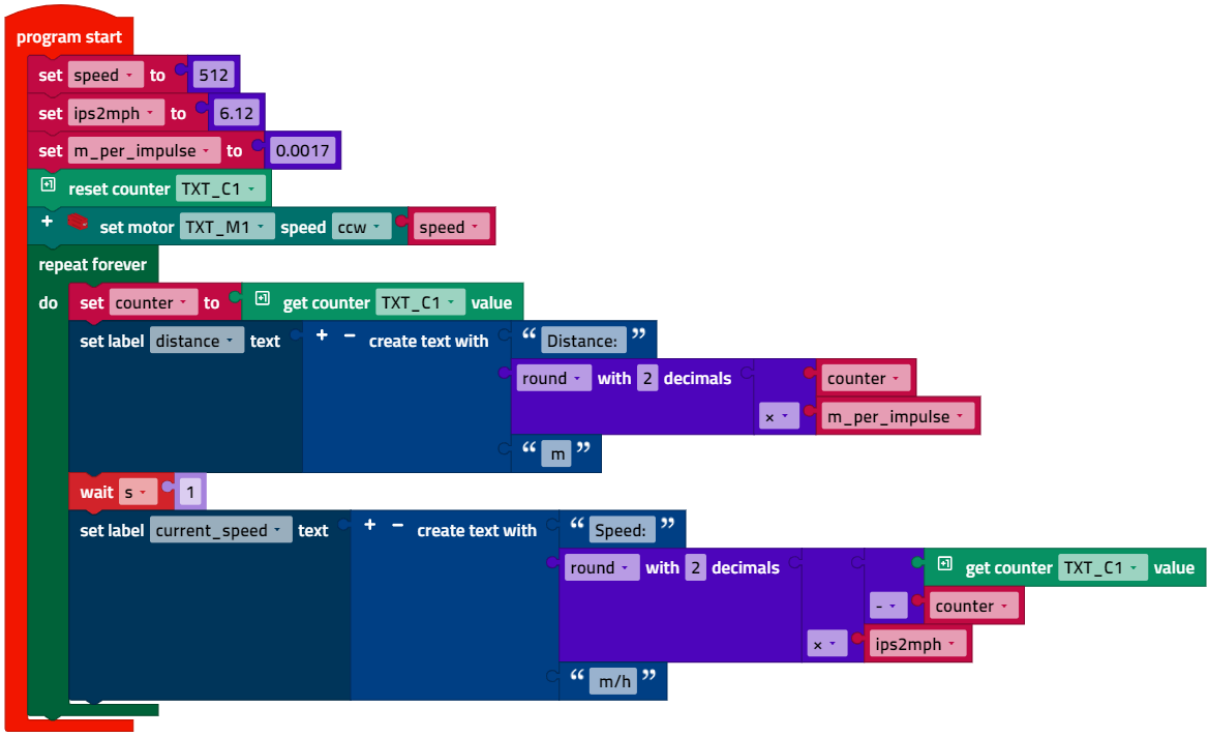


Programm (Beispiel) Tachometer:



Tachometer.ft

Programm (Beispiel) Hodometer und Tachometer:

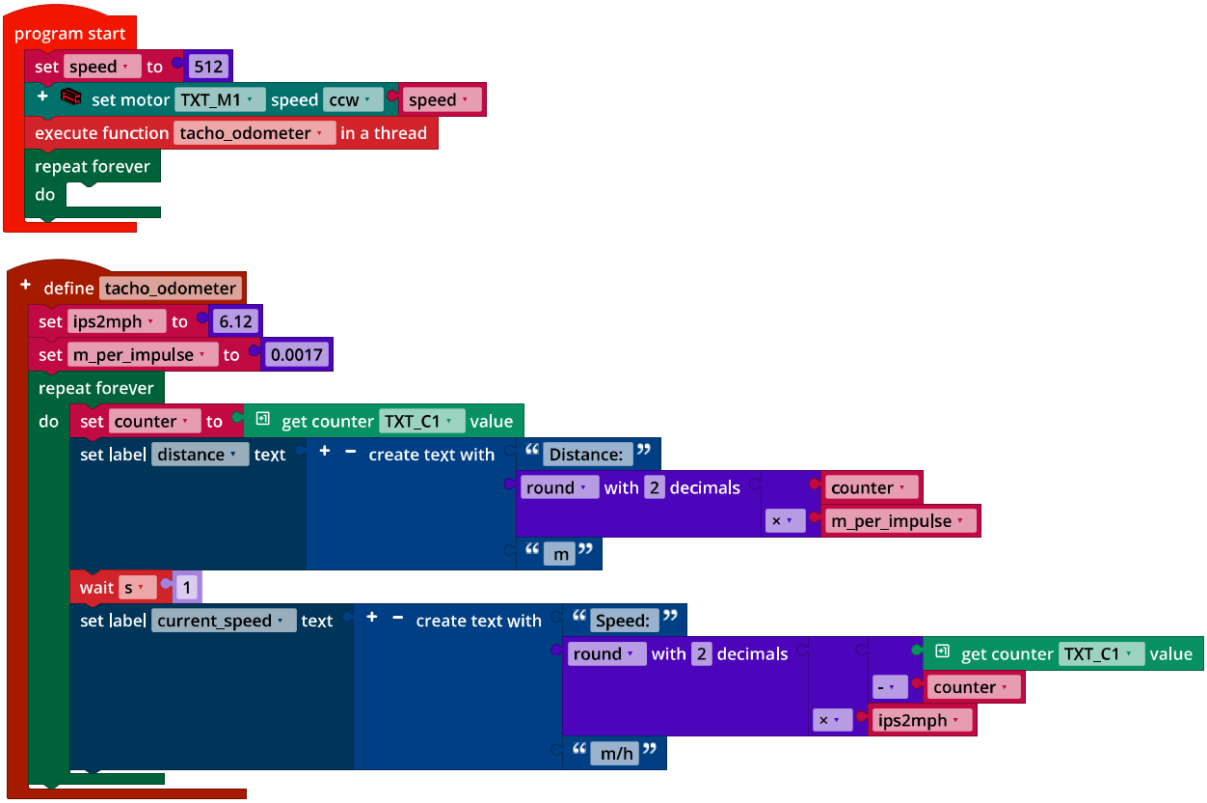


Tachometer_and_Odometer.ft

Experimentieraufgaben

1. Tachometer und Hodometer nebenläufig

Programm (Beispiel):

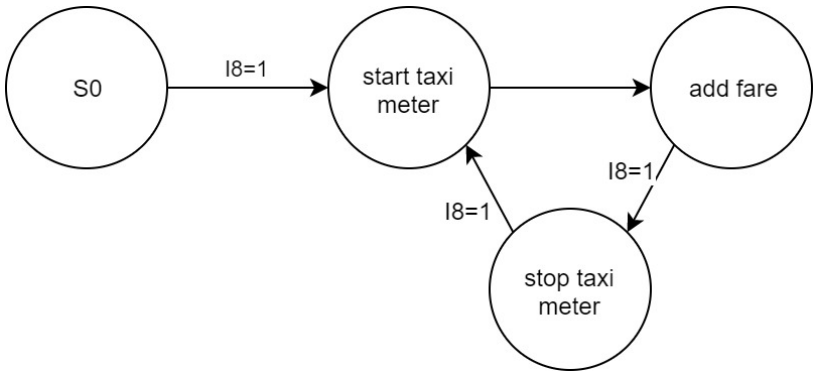


Tachometer_and_Odometer_Thread.ft

2. Taxameter

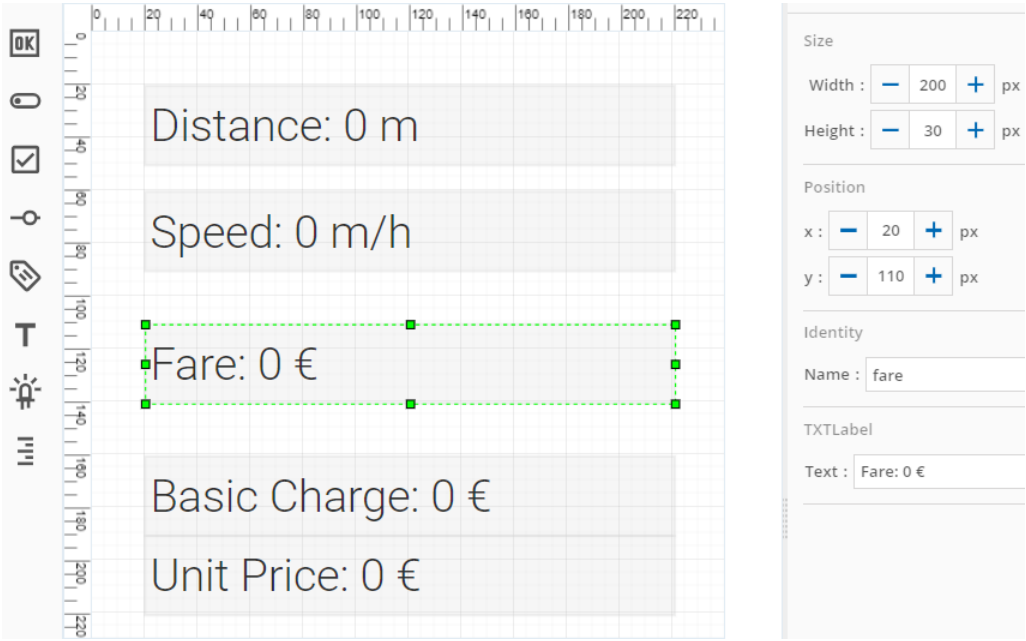
2a. Da das Größenverhältnis des Fahrzeugmodells zu einem echten Fahrzeug bei etwa 1:10 liegt, bietet sich bei einem Taxameter eine Preisangabe für 100-m-Strecken an, z. B. 2 €/100 m und 3,50 € Grundgebühr.

2b. Zustandsübergangsdiagramm:



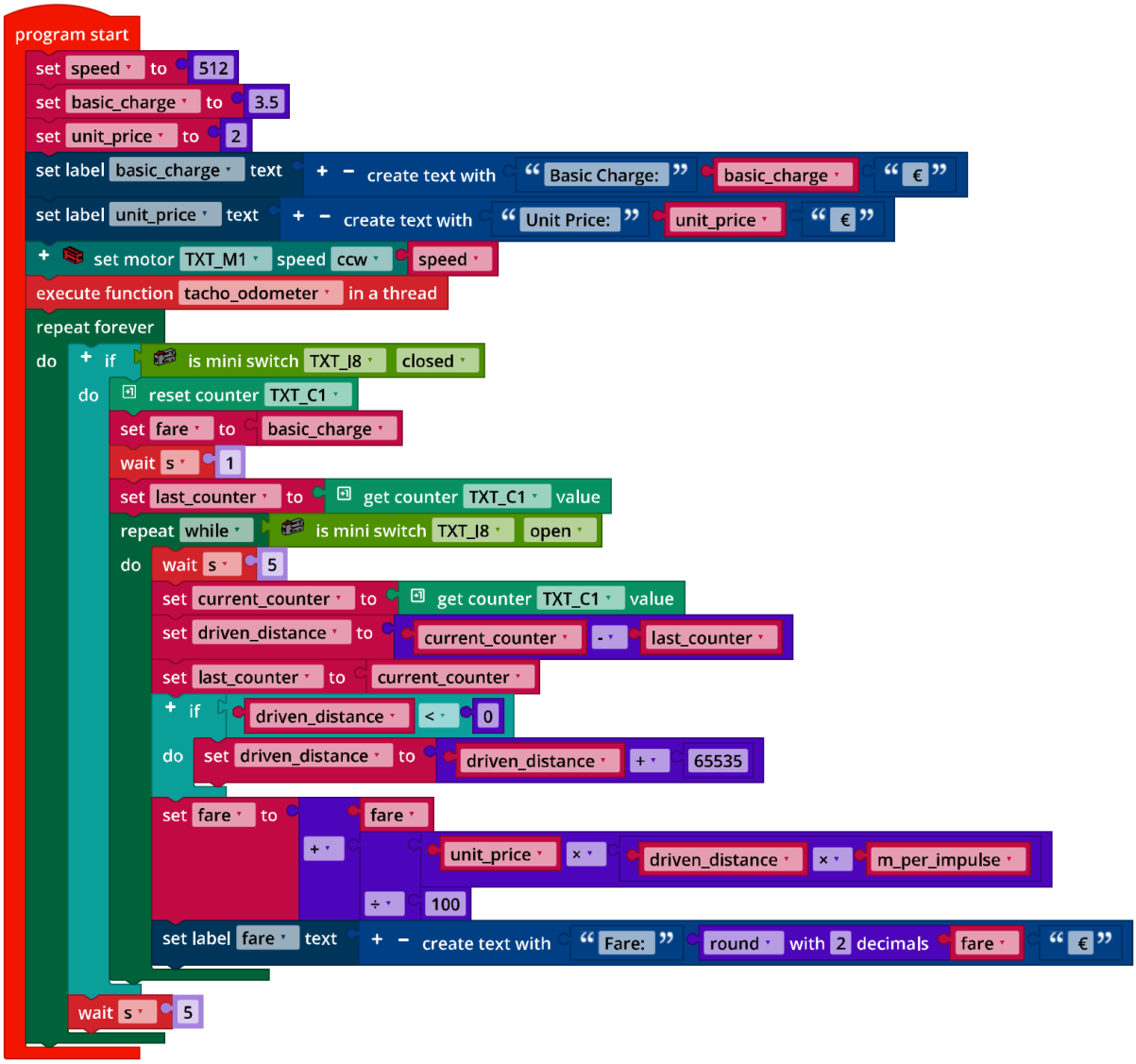
State-Transition_Diagram_Taxi_Meter.drawio

2c. Display-Konfiguration:



Die Encoder-Impulse werden nur bis 65.535 gezählt. Deshalb sollte der Fahrpreis regelmäßig aufaddiert werden, z. B. einmal alle fünf Sekunden.

Programmauszug (Beispiel) Taxameter:



Taxi_Meter.ft

Auch das Taxameter kann man in einen nebenläufigen Prozess (Thread) verschieben. Es läuft dann wie das Tacho- und das Hodometer einfach „mit“, während die Navigation des Fahrzeugs im Hauptprogramm erfolgt.

Anlagen

Aufgabe 1: Tachometer, Hodometer und Taxameter

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Maßband oder Zollstock, Papierstreifen (für die Messung des Reifenumfangs).

Weiterführende Informationen

[1] Andreas Wolf: *Tachometer. Die Geschichte eines unverzichtbaren Instruments*. 04.08.2014.

[2] Alper Aribal (SeoRocket): *Taxameter*. DeWiki.de.

[3] Online-Diagrammeditor zur Erstellung von Zustandsübergangsdiagrammen (Format drawio): https://www.diagrammeditor.de/

Name: _____ Klasse: ___ Datum: _____

Aufgabe 2

Bremsassistent, Tempomat und Spurhalteassistent

Rechtzeitiges Bremsen vor einem Hindernis und eigenständiges Halten der Fahrspur und der Geschwindigkeit sind wichtige Fahrerassistenzsysteme, die in dieser Aufgabe entwickelt werden. Damit das Fahrzeug bei deinen Tests nicht unkontrolliert davonfährt, wird es zunächst mit einem „Not-Halt“ ausgestattet.

Konstruktionsaufgabe

Für diese Aufgabe benötigen wir den Servo-Motor, den Ultraschall-Sensor und die USB-Kamera. Schließe den Ultraschall-Sensor an I1, den Servo-Motor an S1 und die USB-Kamera an USB1 an (siehe Schaltplan). Beim Servo-Stecker auf die richtige Polung achten: das braune Kabel links, das orangene rechts.

Wichtig: Wenn der TXT gestartet ist, stellt sich der Servo automatisch auf „Geradeaus-Stellung“. Stecke den Servo-Hebel so auf, dass beide Vorderräder in dieser Position einigermaßen genau nach vorne ausgerichtet sind.

Achtung: Wenn du den Servo bei eingeschaltetem TXT mit der Hand bewegst, kannst du ihn beschädigen. Auch darf der Servo keine Position ansteuern, die jenseits des Anschlags des Servo-Hebels liegt – daher sollten für Servo-Befehle nur Positionen zwischen etwa 100 und 400 genutzt werden. Der Lenkbereich unseres Fahrzeugs ist noch etwas kleiner; er liegt ungefähr bei Servo-Werten zwischen 130 und 370.

Mit dem Interface-Test kannst du den Bereich des Lenkeinschlags, in dem sich die Vorderräder noch frei drehen, sehr einfach ausmessen.

Wir werden das Fahrzeug nun mit einem Bremsassistenten, einem Tempomaten und einem Spurhalteassistenten ausstatten.

Programmieraufgaben

1. Not-Halt

Wir beginnen mit einem „Not-Halt“-Mechanismus, bevor wir das Fahrzeug autonom fahren lassen: Bei einem lauten Klatschen soll das Fahrzeug sofort anhalten.

Programmiere den „Not-Halt“ als nebenläufigen Prozess (Thread), der über das Mikrofon der Kamera gesteuert wird. Eine passende Lautstärke-Schwelle, bei der der Motor stoppen soll, kannst du experimentell bestimmen, indem du dir den vom Mikrofon bestimmten Lautstärke-Wert zunächst auf der Konsole anzeigen lässt.

Beachte: Der Motor erzeugt selbst ein relativ lautes Geräusch.

2. Geradeausfahrt

Du wirst feststellen, dass das Fahrzeug nicht exakt geradeaus fährt. Bevor wir in den Experimentieraufgaben die Lenkung automatisch an der Fahrspur ausrichten, sollst du zunächst versuchen, den Servo einigermaßen auf Geradeausfahrt zu stellen.

Eine erste Näherung erhälst du mit dem Interface-Test. Testen musst du das Ergebnis jedoch experimentell: Schreibe ein Blockly-Testprogramm und korrigiere den Wert für die initiale Servo-Einstellung so lange, bis das Fahrzeug über eine Teststrecke von 2 m möglichst exakt die Richtung hält.

Gib‘ in den folgenden Programmen die korrekte Servo-Einstellung in einer Variablen vor und richte den Servo zu Programmbeginn mit diesem Wert aus.

3. Bremsassistent

Ein Bremsassistenzsystem soll nun dafür sorgen, dass dein autonom fahrendes Fahrzeug nicht auf ein stehendes (oder auch bewegtes) Hindernis auffährt, wenn der Fahrer unaufmerksam ist oder zu spät reagiert. Dazu soll rechtzeitig eine Bremsung eingeleitet werden, sodass das Fahrzeug dem Hindernis nicht näher als 10 cm kommt.

Den Abstand zu einem Hindernis (oder zu einem vorausfahrenden Fahrzeug) kannst du mit dem Ultraschall-Sensor messen, den du bereits aus dem Robotics TXT 4.0 Base Set kennst.

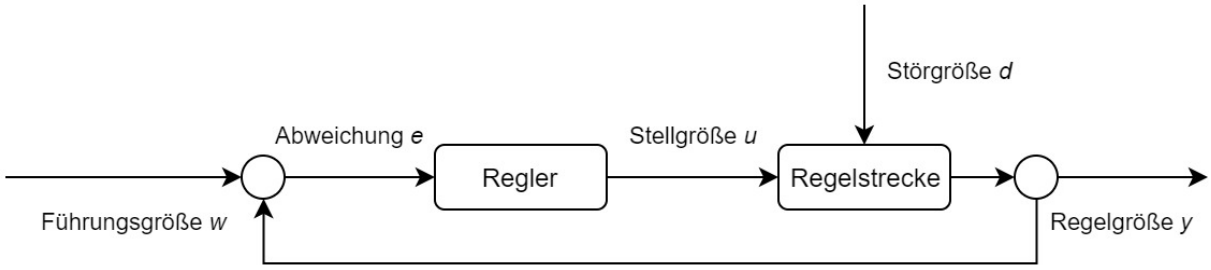
Schreibe ein entsprechendes Blockly-Programm und teste es mit verschiedenen Hindernissen. Nutze den „Not-Halt“-Thread aus der Programmieraufgabe 1, um das Fahrzeug erforderlichenfalls durch Klatschen stoppen zu können.

Beachte: Dein Fahrzeug legt auch während der Abstandsmessung eine Fahrstrecke zurück.

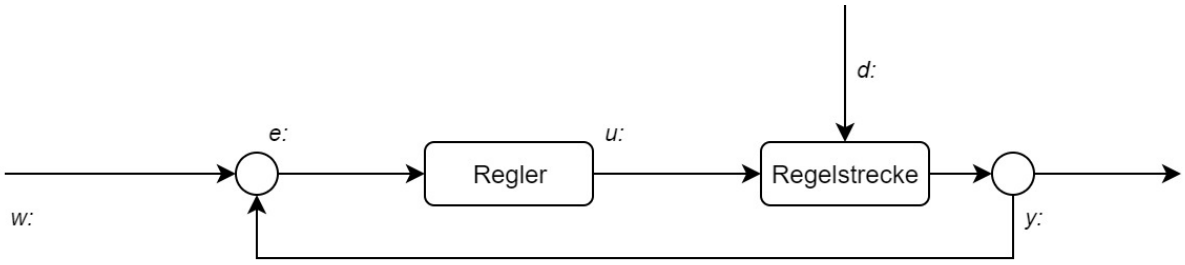
4. Tempomat

Ein Fahrerassistenzsystem, das heute in den meisten Mittelklassewagen enthalten ist, ist der *Tempomat*: Ein Geschwindigkeitsregler, der dafür sorgt, dass ein Fahrzeug eine vorgegebene Geschwindigkeit einhält [1].

Betrachte dazu zunächst die folgende Darstellung eines Regelkreises. Du kennst sie bereits aus Aufgabe 8 des Robotics TXT 4.0 Base Set.



4a. Welche Größen entsprechen beim Tempomaten den Parametern *w*, *e*, *u*, *d* und *y*? Beschrifte den folgenden Regelkreis entsprechend:



4b. Programmiere eine Tempomat-Regelung in Blockly. Als Führungsgröße soll die gewünschte Geschwindigkeit in m/h in einer Variablen vorgegeben sein.

Tipp: Mit dem „Map“-Kommando kannst du die Führungsgröße sehr elegant in die benötigte Motorspannung umrechnen, wenn du die Geschwindigkeit des Fahrzeugs in m/h kennst.

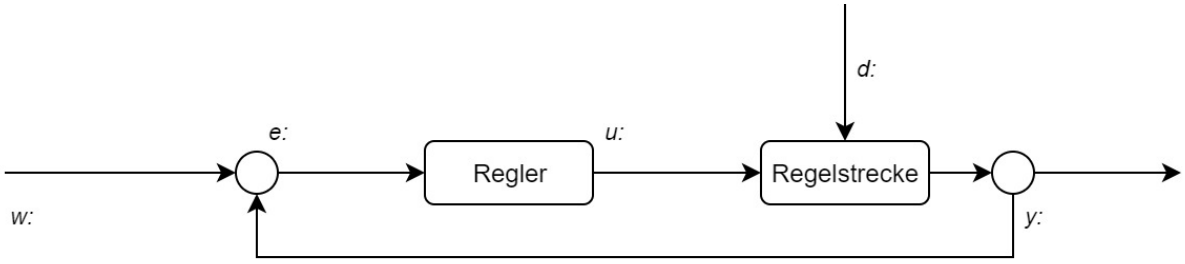
Beachte: Bei einen „Not-Halt“ und auch bei einem Halt vor einem Hindernis darf der Tempomat die Geschwindigkeit anschließend nicht wieder hochsetzen. Passe diese beiden Threads in deinem Programm entsprechend an.

Experimentieraufgabe

1. Spurhalteassistent mit P-Regler

Jetzt soll das Fahrzeug lernen, selbstständig einer Fahrspur zu folgen. Dazu soll es sich mit Hilfe der Kamera an der rechten Fahrbahnbegrenzung orientieren. Auch dieses Fahrerassistenzsystem sollst du mit einem Proportionalregler (P-Regler) verwirklichen.

1a. Beschrifte zunächst den folgenden Regelkreis:



1b. Erweitere dein Hauptprogramm nun um einen entsprechenden Proportional-Regler, der die Servo-Lenkung abhängig von der Spurabweichung korrigiert.

Tipp: Konfiguriere die Linienerkennung der Kamera so, dass sie die Position „0“ liefert, wenn das Fahrzeug in 2 cm Abstand parallel zur rechten Fahrbahnbegrenzung steht.

Tipp: Begrenze die Geschwindigkeit deines Fahrzeugs auf 350 und teste den P-Regler zunächst mit einem sehr kleinen Wert ($k_p = 0{,}1$). Erhöhe den Proportionalitätsfaktor so lange, bis der Regler „aufschwingt“, und wähle dann den Wert, bei dem der Regler sich am schnellsten „einschwingt“.

Verwende zum Testen deines Reglers den geraden Straßenabschnitt auf dem beiliegenden Bogen.

1c. Ergänze Dein Programm um eine Textausgabe, die nach jeder Änderung der Position der erkannten Fahrbahnbegrenzung

- die Zeit (in ms), die seit dem Start des Programms verstrichen ist, und
- den Wert der aktuellen Position

durch ein Leerzeichen getrennt auf der Konsole ausgibt.

Kopiere nach jeder Testfahrt mit einem anderen Wert für k_p die Konsolen-Ausgaben in eine Tabellenkalkulation und lass‘ dir die Werte grafisch in einem Diagramm (x: Zeit, y: Position) anzeigen. Wähle den Proportionalitätsfaktor k_p, bei dem sich der Kurvenverlauf am schnellsten „einschwingt“.

2. Spurhalteassistent mit PD-Regler

Wie beim Buggy in Aufgabe 8 des Robotics TXT 4.0 Base Set kannst du durch eine Erweiterung des Reglers um ein „D“-Glied (Differential-Anteil), das die Größe der Veränderung der Abweichung der erkannten Fahrbahnbegrenzung von der Position „0“ bei der Lenkkorrektur berücksichtigt, das Überschwingen weiter dämpfen.

2a. Erweitere den P-Regler aus Experimentieraufgabe 1 zu einem PD-Regler.

2b. Führe Testfahrten mit unterschiedlichen Werten für den Differential-Faktor k_d durch und lass‘ dir die Daten in einem Tabellenkalkulationsprogramm grafisch anzeigen.

Tipp: Beginne deine Tests mit dem Differential-Faktor $k_d = 0{,}01$ und erhöhe dessen Wert so lange in Schritten von 0,005, bis das Überschwingen des P-Reglers gut gedämpft wird.

2c. Wenn du mehrere gerade und gekrümmte Straßenabschnitte aneinander legst, kannst du den Spurhalteassistenten auf einem herausfordernden Straßenparcours testen.

Anlagen

Aufgabe 2: Bremsassistent, Tempomat und Spurhalteassistent

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Fahrbahn mit Markierungen auf beiliegendem Bogen (oder als Ausdruck der Fahrbahn-Datei)
- Hindernis (z. B. ein Buch oder ein Karton)

Weiterführende Informationen

[1] Jim Meininghaus: *Die Geschichte des Tempomaten. Wie ein Blinder das Autofahren veränderte*. 03.03.2014, motor-talk.de.

[2] Thomas Paulsen: *Autonomes Fahren: Die 5 Stufen zum selbstfahrenden Auto*. 07.11.2018, adac.de.

[3] Online-Diagrammeditor zur Erstellung von Zustandsübergangsdiagrammen (Format drawio): https://www.diagrammeditor.de/

Aufgabe 2: Bremsassistent, Tempomat und Spurhalteassistent

Das Fahrzeug wird nun mit ersten autonomen Fähigkeiten (Fahrerassistenzsystemen) ausgestattet. So soll es beim Auftauchen eines Hindernisses eine Bremsung einleiten (Bremsassistent), eine vorgegebene Geschwindigkeit beibehalten (Tempomat) und selbstständig einer Fahrspur folgen (Spurhalteassistent).

Thema

Fahrerassistenzsysteme (Bremsassistent, Spurhalteassistent und Tempomat), Sensorik (Auswertung von Encoderimpulsen, Ultraschall-Werten und Kamerabildern zur Geschwindigkeitskontrolle, Erkennung von Hindernissen und Fahrspuren) und Regler (P- und PD-Regler).

Lernziele

- Abstandsmessung mit Ultraschall-Sensor
- Analoge Proportionalregelung mit Geschwindigkeitsmessung (Tempomat) und unter Verwendung einer Kamera mit Bildauswertung (Spurhalteassistent)
- PD-Regler (Spurhalteassistent)

Zeitaufwand

Am autonomen Fahrzeug werden zusätzlich der Servo-Motor und die USB-Kamera angeschlossen.

Die Aufgabe baut auf dem „Hinderniserkenner“ (Ultraschall-Abstandsmessung) und dem „Spurfolger“ (Proportionalregler) aus den Aufgaben des Robotics TXT 4.0 Base Set auf. Für die Entwicklung der Programme zur Lösung der Programmieraufgaben benötigen Schülerinnen und Schüler je nach Alter und Vorkenntnissen 180-270 Minuten (vier bis sechs Unterrichtsstunden). Das Funktionsprinzip eines Proportional-reglers sollte zuvor im Unterricht behandelt worden sein.

Die Experimentieraufgaben (Spurhalteassistent mit P- und PD-Regler) können besonders begabten oder älteren Schülerinnen und Schülern gestellt werden. Ihre Lösung erfordert weitere ca. 90-180 Minuten (zwei bis vier Unterrichtsstunden).

Bezug Curriculum

Land	Stufe/Fächer	Bezüge
BW	SEK I	GYM 8/9/10 NWT-3.2.4.3 Steuerungsabläufe (Ampelsteuerung, Robotik) (7), Informationsverarbeitung - Autonomes Fahren (8), S.27; IMP 8-3.1.1.2 Algorithmen (1), S. 28ff; INFWF 8-3.1.2 Algorithmen (1), S. 15; INFWF 9-3.2.2 Algorithmen (2), S. 21; INFWF 10-3.3.2 Algorithmen (2), S. 28;
BY	SEK I	RS- IT 2.7 Logik und Robotik, S.699; GYM 9/10 LPLUS INF - Modellieren, Implementieren, Anwenden, Softwareprojekte
BE	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
BB	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
HB	SEK II	GYM OS INF-Algorithmen und Datenstrukturen, S. 6; GYM OS INF-Imperative Programmierung, S. 7
HH	SEK I	Stadtteil 9/10 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 20; GYM 9 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 19
HE	SEK I	ohne curricularen Vorgaben
MV	SEK I	GYM 5 INF-3 Programmieren? Kinderleicht!, S.16; GYM 6 INF-3 Entscheidungen treffen und Spiele gestalten, S.19 GYM 7 INF-3 Spiele entwickeln, S.22; GYM 8 INF-3 Sensorgesteuerte Anwendungen entwickeln, S.25; GYM 9 INF-3 Problemlösen durch Programmieren, S.32
NI	SEK I	KC-INF LF Algorithmisches Problemlösen; S.19; KC-INF LF Automatisierte Prozesse, S.22; SEK 2 KC-INF LF1 Algorithmen und Datenstrukturen, S.14; SEK 2 KC-INF LF1 Informationen und Daten, S.16; ; SEK 2 KC-INF LF1 Automaten und Sprachen, S.19
NW	SEK I, II	RS 9/10 WPF TECHNIK 2.3 Inhaltsfeld 7: Kommunikations- und Digitaltechnik S.23; 5/6 KLP INF - Algorithmen, S. 17, 18; 5/6 KLP INF - Automaten und künstliche Intelligenz, S. 18; SEK 2 KLP GOS INF - 2 Algorithmen, S. 21 ff; KLP GOS INF - 3 Formale Sprachen und Automaten, S. 22
RP	SEK I	IPS 5 INF - Informatiksysteme und Netze, S. 7; IGS/GYM INF-2.1 Grundlagen der Informationsverarbeitung, S. 17; IGS/GYM INF-2.2 Algorithmisches Problemlösen, S. 20
SL	SEK I, II	GYM 9 INF - Imperative Programmierung, S. 3; INF - Algorithmik, S. 3; GYM OS INF GOS-Funktionsweise von Computersystemen, S.9ff.

SN	SEK I	GYM 7 INF LB 3: Computer verwenden – Komplexaufgabe, S. 7; GYM 8 INF LB 2: Daten verarbeiten, S.10
ST	SEK I, II	GYM 9 INF 3.2 Algorithmen interpretieren und entwickeln, S.15 ff.; GYM 11/12 INF 3.4 Kurs 3 Software Engineering und Projektarbeit, S. 23
SH	SEK I	INF PB1 Modellieren und Strukturieren, S. 12; INF PB2 Implementieren, Programmieren, Realisieren, S. 13; ; FA Physik, Variabilität S.13
TH	SEK I	GYM 10 INF - 2.3 Algorithmen, S. 14 ff.; GYM 10 INF 2.5.1 Technische Informatik, S. 18ff.

Anlagen
Aufgabe 2: Bremsassistent, Tempomat und Spurhalteassistent

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Fahrbahn mit Markierungen auf beiliegendem Bogen (oder als Ausdruck der Fahrbahn-Datei)
- Hindernis (z. B. ein Buch oder ein Karton)

Weiterführende Informationen

[1] Jim Meininghaus: *Die Geschichte des Tempomaten. Wie ein Blinder das Autofahren veränderte*. 03.03.2014, motor-talk.de.

[2] Thomas Paulsen: *Autonomes Fahren: Die 5 Stufen zum selbstfahrenden Auto*. 07.11.2018, adac.de.

[3] Online-Diagrammeditor zur Erstellung von Zustandsübergangsdiagrammen (Format drawio): https://www.diagrammeditor.de/

Name: _____ Klasse: ___ Datum: _____

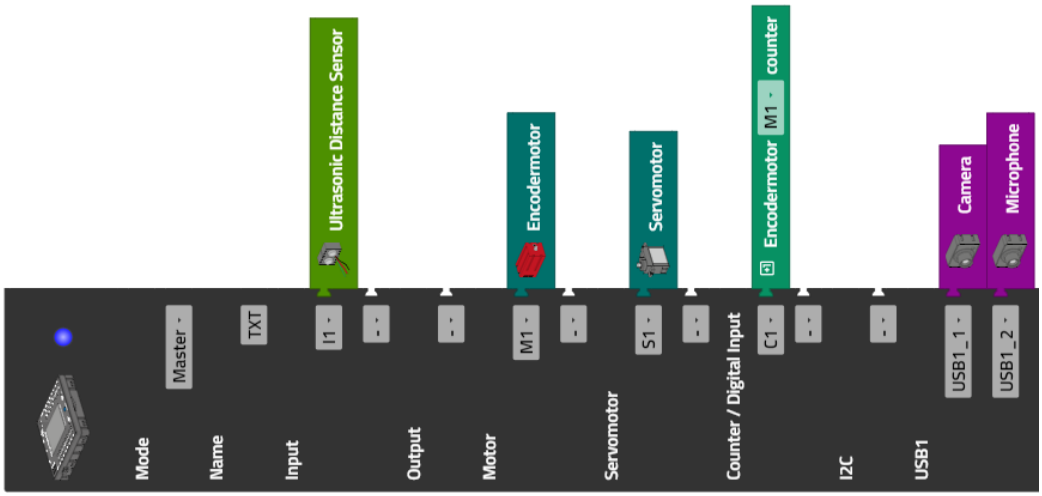
Lösungsblatt Aufgabe 2

Bremsassistent, Tempomat und Spurhalteassistent

Bei der Programmierung sollten die Schülerinnen und Schüler regen Gebrauch von der Ausgabe von Variablen auf der Konsole machen und Tests zunächst mit einem „aufgebockten" Fahrzeug (oder ohne montierte Hinterräder) durchführen. Die „Live"-Tests können dann mit zu einem Parcours zusammengelegten Fahrbahnabschnitten durchgeführt werden. Zum Ausdrucken zusätzlicher Fahrbahnabschnitte gibt es die Fahrbahnen als pdf-Datei zum Download.

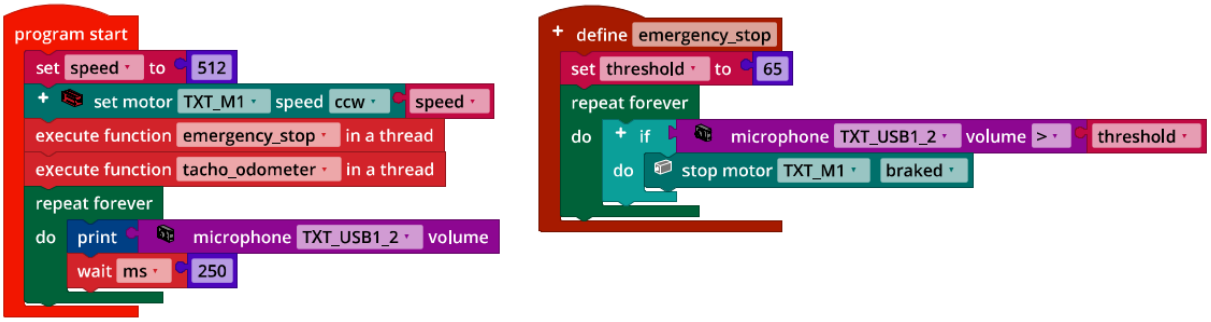
Programmieraufgaben

Konfiguration der Aktoren und Sensoren:



1. Not-Halt

Programmauszug (Beispiel): Not-Halt des Fahrzeugs bei einem lauten Geräusch (z. B. Klatschen); Ausgabe der Lautstärke auf der Konsole:

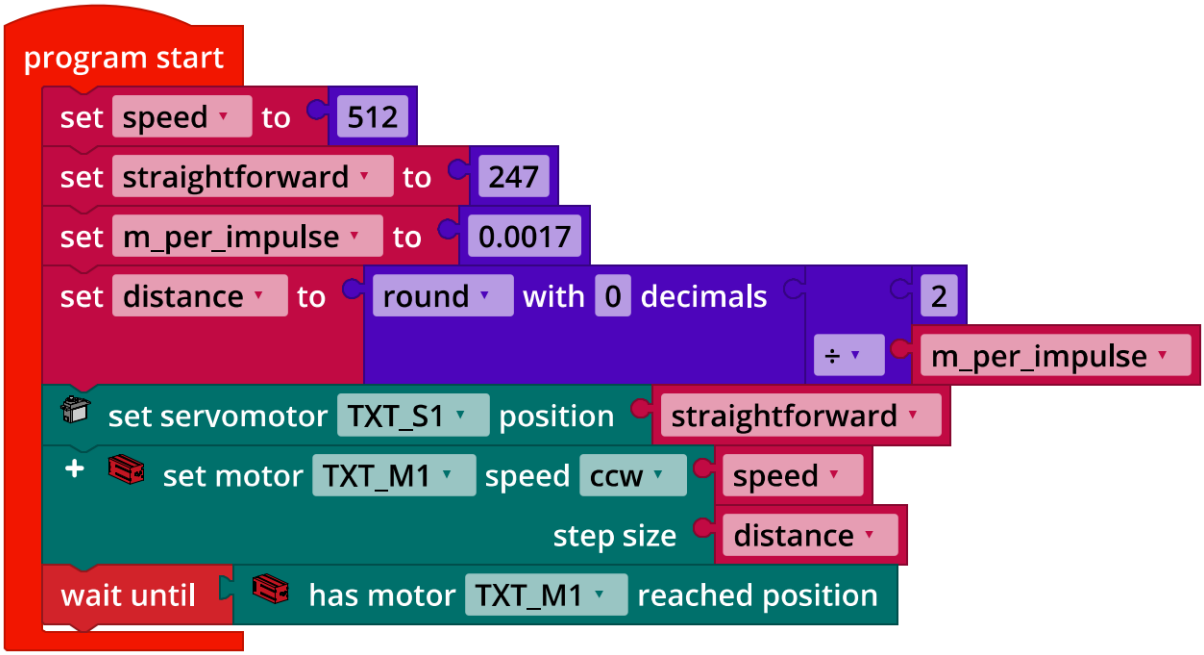


Emergency_Stop.ft

2. Geradeausfahrt

Die Einstellung des Servos für die Geradeausfahrt unterscheidet sich von Modell zu Modell. Es kann erforderlich sein, den Wert von Zeit zu Zeit zu überprüfen und ggf. nachzujustieren. Im folgenden Programmbeispiel für den (experimentellen) Test der Geradeausfahrt wird er in der Variablen „straightforward“ vorgegeben.

Programm (Beispiel):



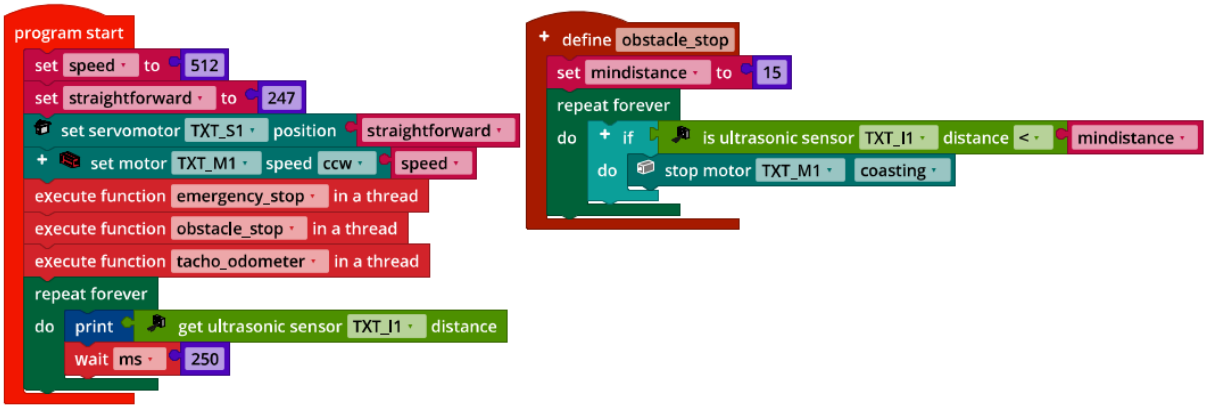
Servo_Calibration.ft

3. Bremsassistenz

Damit das Fahrzeug rechtzeitig eine Bremsung einleitet, muss der Schwellenwert für den Abstand, ab dem der Bremsvorgang eingeleitet werden soll, über 10 cm liegen (Reaktionszeit der Ultraschallmessung, Bremsweg). Den Bremsweg kann man simulieren, indem man den Motor im Modus „coasting“ stoppt.

Um den passenden Abstands-Schwellenwert für den Beginn des Bremsvorgangs zu bestimmen, lässt man das Fahrzeug mit Höchstgeschwindigkeit auf ein stehendes Hindernis zufahren und stoppen. Den Schwellenwert passt man dann so lange an, bis das Fahrzeug 10 cm vor dem Hindernis zum Stehen kommt.

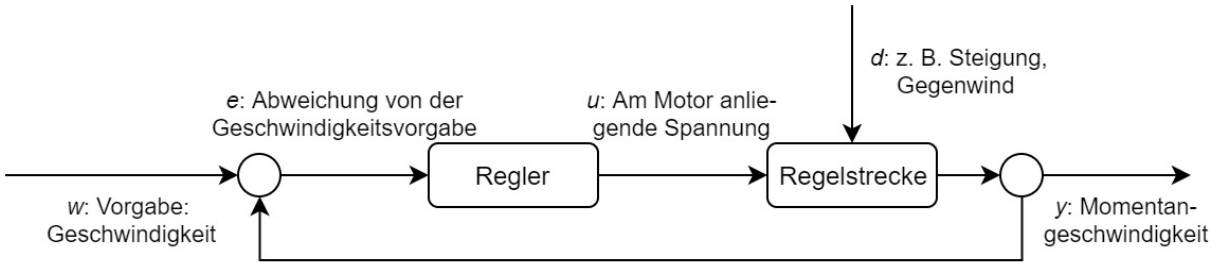
Programmauszug (Beispiel) Bremsassistenzsystem:



Brake_Assist.ft

4. Tempomat

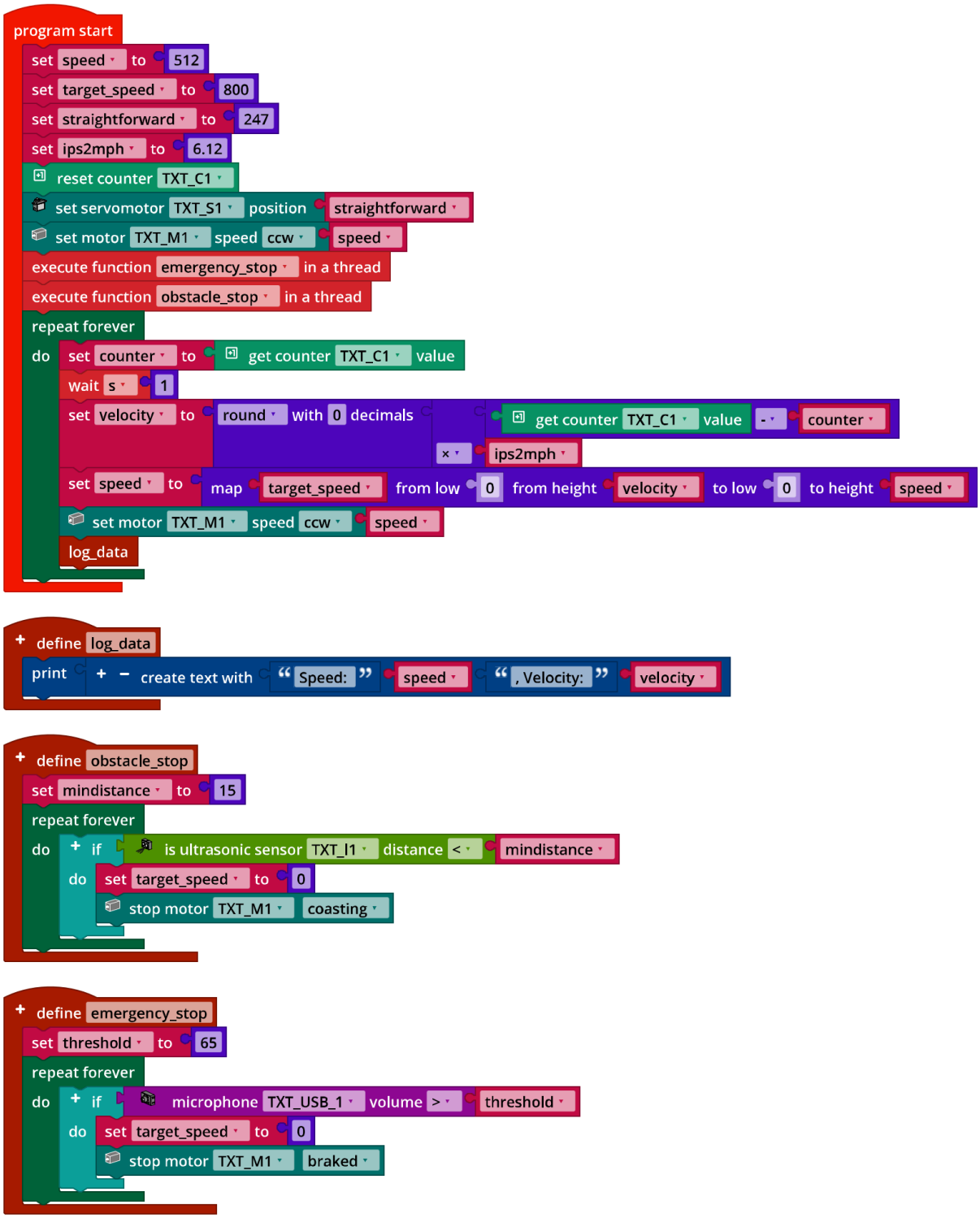
4a. Regelkreis des Tempomaten:



Regelkreis_Tempomat.drawio

Der Tempomat hält die Geschwindigkeit auch bei einer Steigung oder anderen Fahrwiderständen. Das kann man im aufgebockten Zustand des Fahrzeugs anschaulich demonstrieren, indem man mit der Hand die Reifen bremst.

4b. Programmauszug (Beispiel) Tempomat:

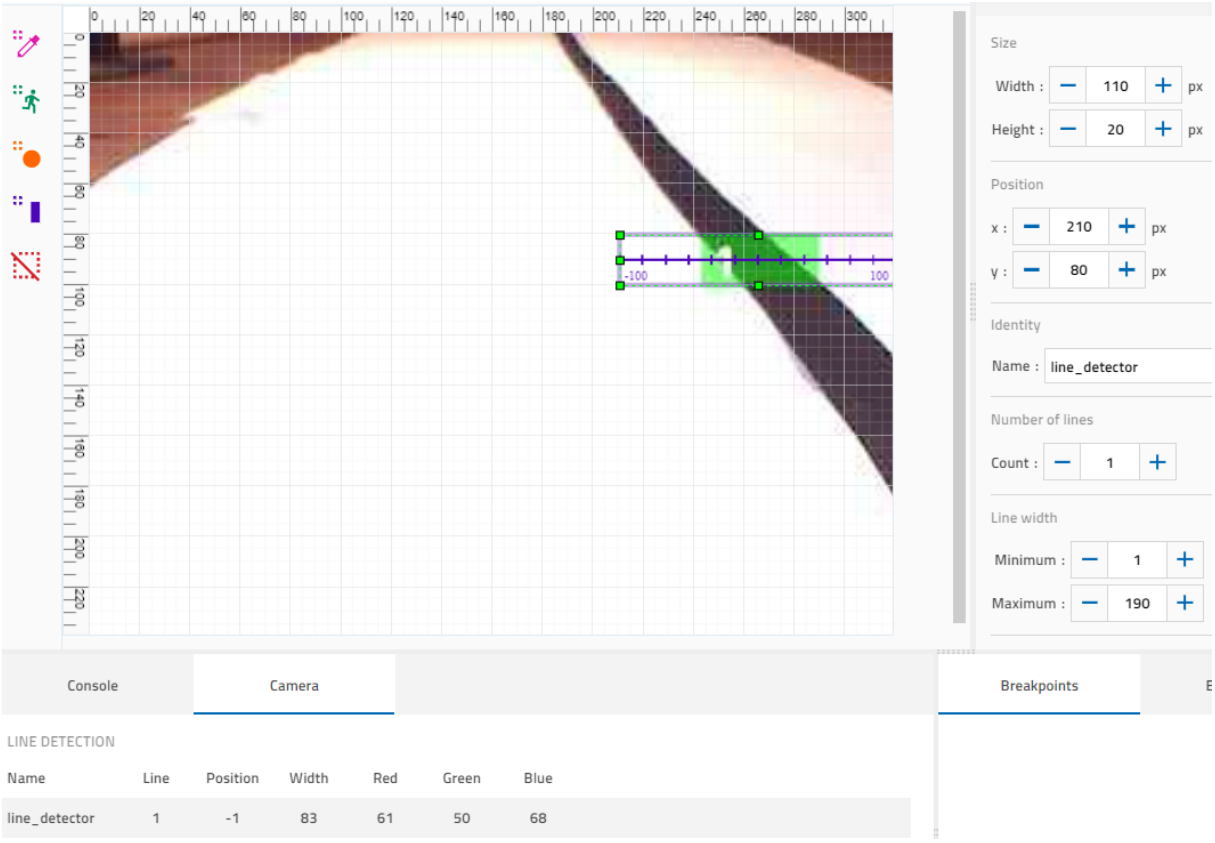


Speed_Control.ft

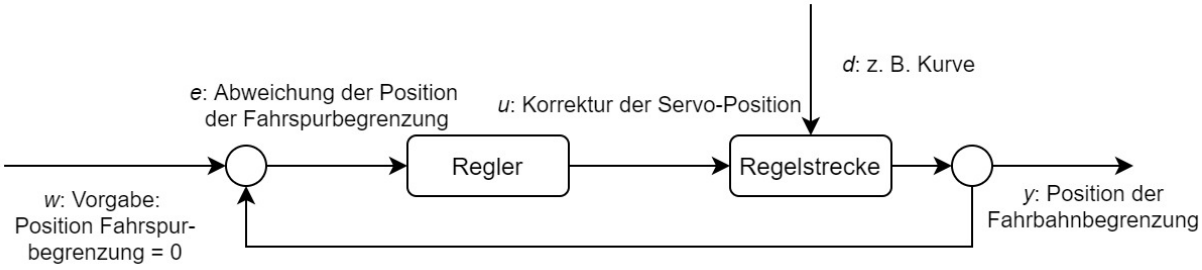
Experimentieraufgaben

1. Spurhalteassistent mit P-Regler

Konfiguration der Kamera (Linienerkennung):

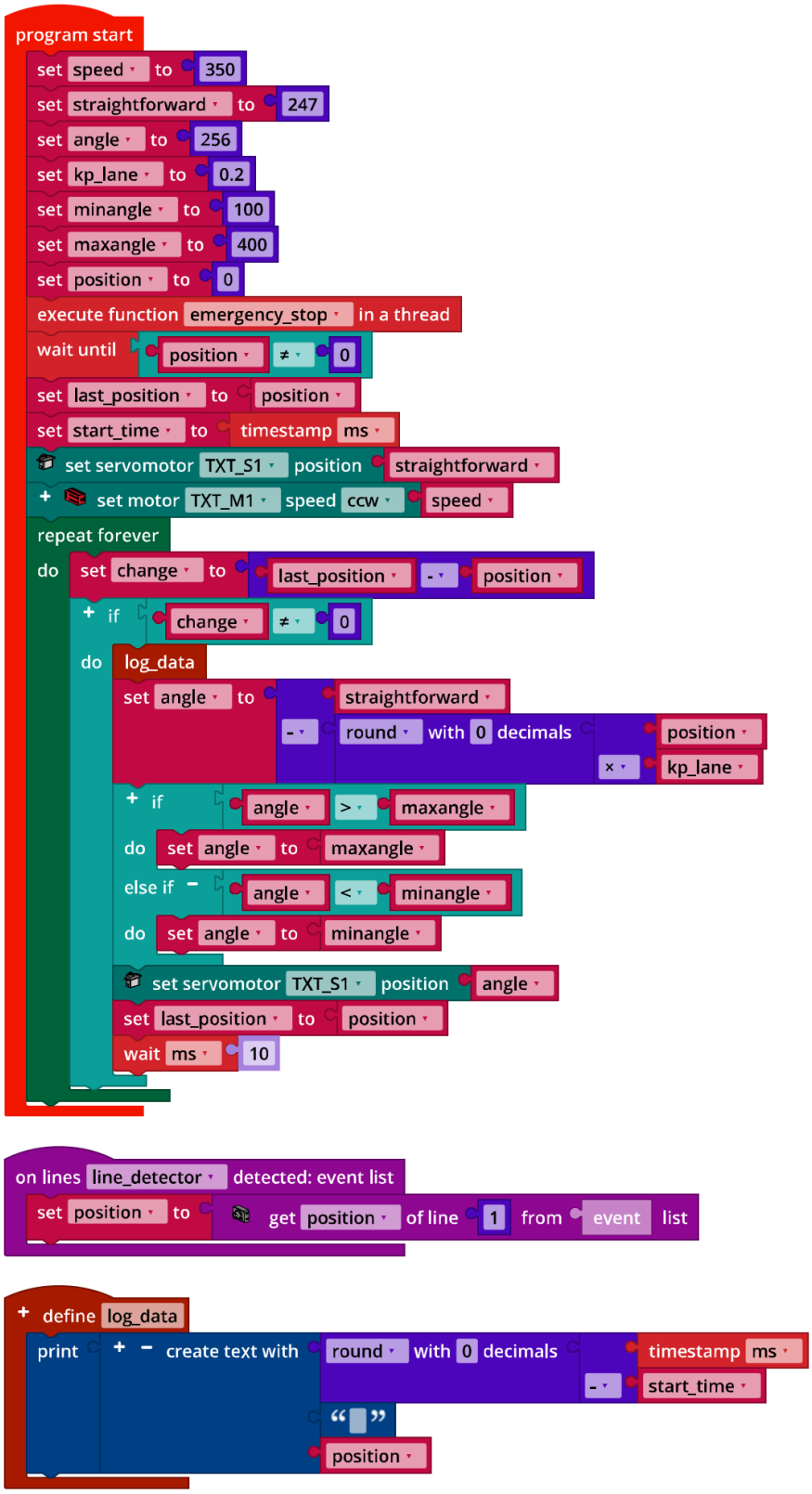


1a. Regelkreis des Spurhalteassistenten:



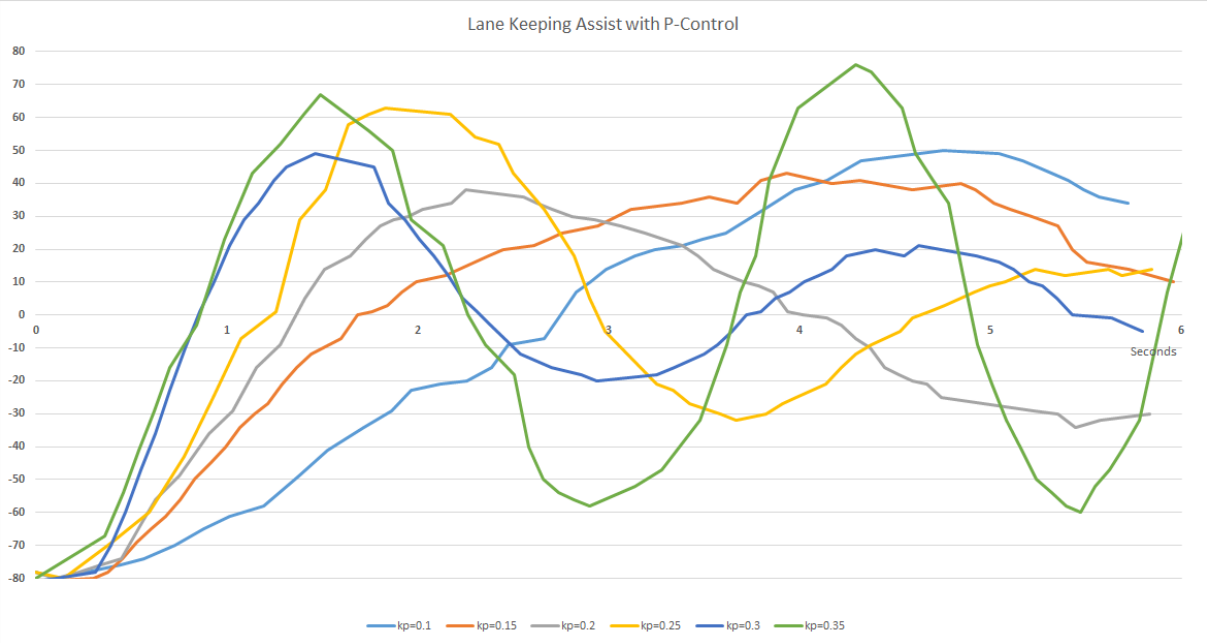
Regelkreis_Spurhalteassistent.drawio

1b. Programm (Beispiel) Spurhalteassistent mit P-Regler:



Lane_Keeping_Assist_P-Control.ft

1c. Messergebnisse des P-Reglers bei einer Geschwindigkeit von 350 (mit $k_p \in \{0.1, 0.15, 0.2, 0.25, 0.3, 0.35\}$):



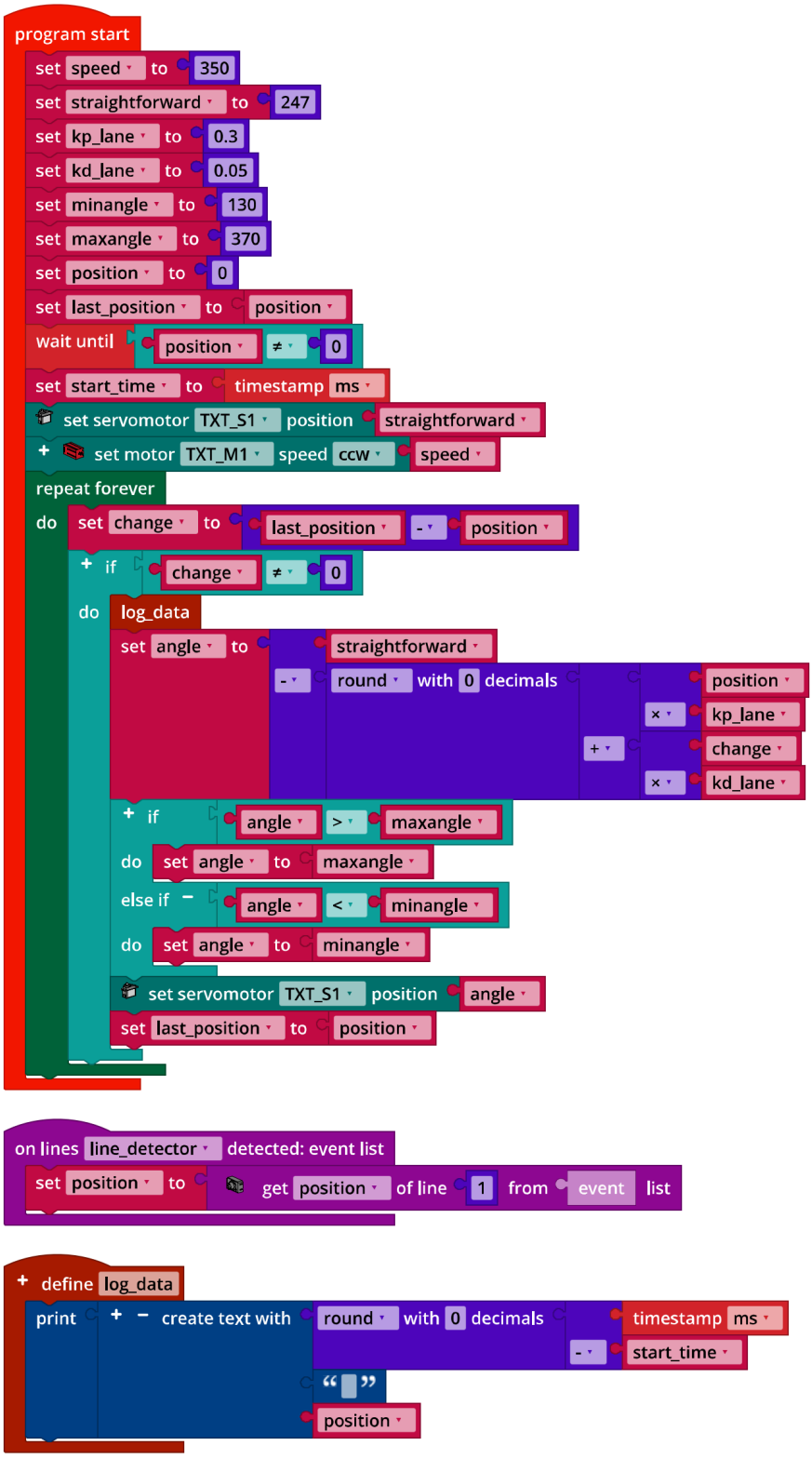
Lane_Keeping_Assist_P-Control_Results.jpg

Die maximale Bildwiederholrate liegt bei 15 fps (*frames per second*), d. h. die Linienerkennung kann höchstens alle etwa 66,7 ms eine neue Positionsbestimmung durchführen.

Bei einem Wert von $k_p = 0.35$ beginnt der Regler in der Beispiellösung zu oszillieren. Mit $k_p = 0.3$ schwingt sich der Regler sehr schnell mit abklingender Amplitude ein. Je nach Modell (Reibung, Motorleistung, …) und Programm können die Messergebnisse der Schülerinnen und Schüler abweichen.

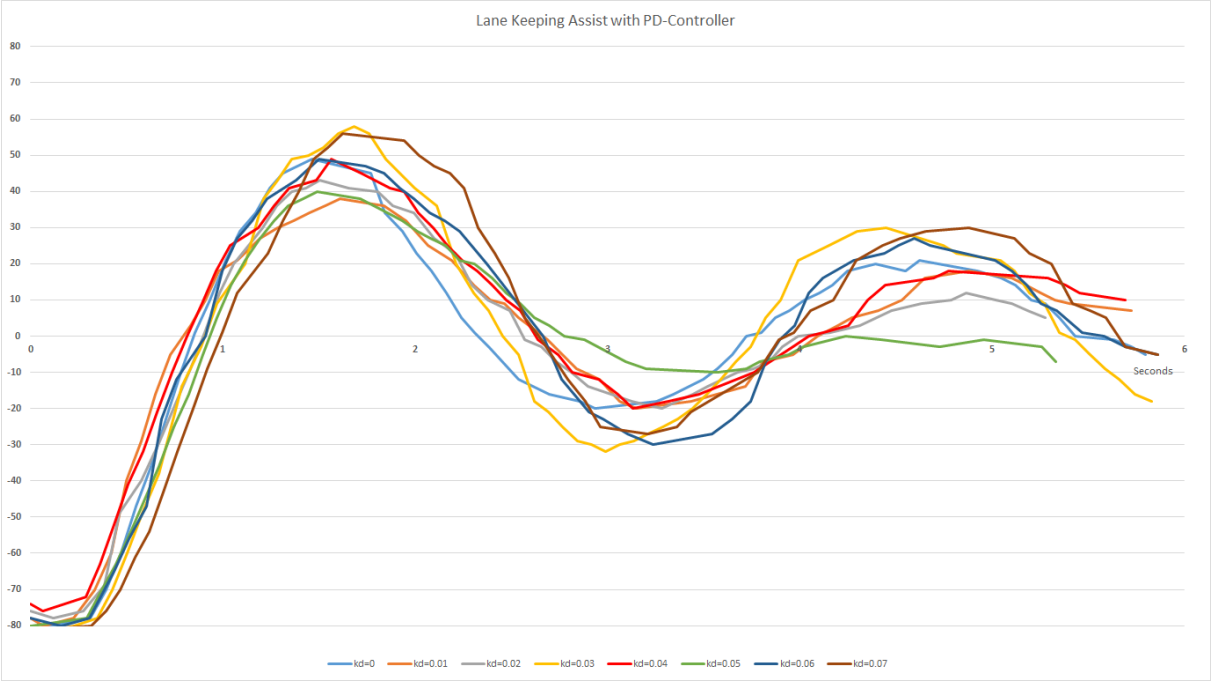
2. Spurhalteassistent mit PD-Regler

2a. Programm (Beispiel):



Lane_Keeping_Assist_PD-Control.ft

2b. Messergebnisse des PD-Reglers bei einer Geschwindigkeit von 350 und mit $k_p = 0.3$ (für $k_d \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07\}$):



Lane_Keeping_Assist_PD-Control_Results.jpg

Die größte Dämpfung wird im Beispielprogramm mit einem Differentialfaktor $k_d = 0.05$ erreicht (grüne Linie in der Abbildung). Auch hier können die Messergebnisse der Schülerinnen und Schüler abweichen; qualitativ sollten die Messverläufe jedoch sehr ähnlich ausfallen.

Anlagen

Aufgabe 2: Bremsassistent, Tempomat und Spurhalteassistent

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Fahrbahn mit Markierungen auf beiliegendem Bogen (oder als Ausdruck der Fahrbahn-Datei)
- Hindernis (z. B. ein Buch oder ein Karton)

Weiterführende Informationen

[1] Jim Meininghaus: *Die Geschichte des Tempomaten. Wie ein Blinder das Autofahren veränderte*. 03.03.2014, motor-talk.de.

[2] Thomas Paulsen: *Autonomes Fahren: Die 5 Stufen zum selbstfahrenden Auto*. 07.11.2018, adac.de.

[3] Online-Diagrammeditor zur Erstellung von Zustandsübergangsdiagrammen (Format drawio): https://www.diagrammeditor.de/

Name: _____ Klasse: ___ Datum: _____

Aufgabe 3

Lichtautomatik

Im Straßenverkehr verursachen die Fahrerassistenzsysteme unseres Fahrzeugs womöglich Unfälle – der nachfolgende Verkehr wird weder durch einen Blinker noch durch ein Bremslicht gewarnt. Daher wird das Fahrzeug nun mit Beleuchtungsfunktionen ausgestattet.

Konstruktionsaufgabe

Schließe die beiden Frontscheinwerfer (Abblendlicht) parallel an den Ausgang O5 und das Rückfahrlicht (weiß) an den Ausgang O4 an. Die Bremsleuchte (rot) wird mit O3 verbunden. Der linke Blinker wird an O7 und der rechte an O8 angeschlossen.

Programmieraufgaben

Dein Steuerungsprogramm mit den Fahrerassistenzsystemen wird nun um automatische Beleuchtungsfunktionen erweitert.

1. Rückfahrlicht

Das Rückfahrlicht soll leuchten, wenn der Motor sich rückwärts dreht. Ergänze dein Steuerungsprogramm um einen entsprechenden Thread.

2. Bremslicht

Das Bremslicht soll leuchten, wenn der Motor verlangsamt.

2a. Ergänze dein Programm um einen entsprechenden Thread.

2b. Damit das Bremslicht auch bei einer kurzen Verzögerung vom nachfolgenden Fahrzeug wahrgenommen wird, soll es 0,5 Sekunden nachleuchten.

2c. Erweitere die Bremslichtfunktion, indem du die Stärke der Verzögerung berücksichtigst: Je größer die Bremswirkung, desto heller soll das Bremslicht aufleuchten.

3. Blinker

Die beiden Blinker sollen automatisch aktiviert werden, wenn der Lenkeinschlag einen bestimmten Winkel überschreitet (hier: Servo-Hebel auf einer Position kleiner 190 bzw.

größer 310). Beachte: Nach Straßenverkehrsordnung muss ein Blinker mit einer Frequenz von 1-2 Hz blinken.

Die Information darüber, ob der Blinker aktiviert ist, soll über ein Semaphor (hier: eine gemeinsam genutzte Status-Variable) zwischen dem Hauptprogramm und dem Thread ausgetauscht werden.

Experimentieraufgaben

1. Warnblinker

Im Falle eines „Not-Halts“ oder einer Bremsung wegen eines Hindernisses soll ein Warnblinklicht aktiviert werden.

Erweitere deinen Blinker-Thread entsprechend.

2. Abblendlicht

Die Frontstrahler und die Rückleuchte sollen automatisch eingeschaltet werden, sobald es dunkel wird (siehe auch Aufgabe 6 des Robotics TXT 4.0 Base Set).

Befestige dazu an deinem Fahrzeug den Fototransistor und schließe ihn an den Eingang I7 an.

Programmiere die Lichtautomatik als weiteren Thread.

Da die Rückleuchte zugleich als Bremslicht genutzt wird, darfst du sie nur mit halber Helligkeit aktivieren. Passe die Bremslichtautomatik so an, dass die Rückleuchte beim Bremsen heller aufleuchtet (siehe Programmieraufgabe 2c oben).

Anlagen

Aufgabe 3: Lichtautomatik

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.

Weiterführende Informationen

[1] Wikipedia: *Nebenläufigkeit*.

Aufgabe 3: Lichtautomatik

In dieser Aufgabe wird das autonome Fahrzeug mit einem automatisch aktivierten Abblendlicht, einem Bremslicht, einem Rückfahrlicht sowie Blinkern und einem Warnblinker ausgestattet.

Thema

Automatisch gesteuerte Fahrzeugbeleuchtung mit zeitabhängigen, nebenläufigen (parallelen) Prozessen (Blinker, Warnblinker) und Verzögerungsschaltungen (Bremslicht).

Lernziel

- Nutzung von Threads zur Programmierung nebenläufiger (quasi-paralleler) Abläufe
- Kommunikation zwischen Threads über Semaphore
- Zeitabhängigkeit von Prozessen (Zeittakt, Verzögerungen)

Zeitaufwand

Am autonomen Fahrzeug sind die sechs LED zu verkabeln. Dafür benötigen die Schülerinnen und Schüler etwa 15-20 Minuten.

Für die Lösung der Programmieraufgaben benötigen Schülerinnen und Schüler ein bis zwei Unterrichtsstunden (45-90 Minuten).

Bezug Curriculum

Land	Stufe/Fächer	Bezüge
BW	SEK I	GYM 8/9/10 NWT-3.2.4.3 Steuerungsabläufe (Ampelsteuerung, Robotik) (7), Informationsverarbeitung - Autonomes Fahren (8), S.27; IMP 8-3.1.1.2 Algorithmen (1), S. 28ff; INFWF 8-3.1.2 Algorithmen (1), S. 15; INFWF 9-3.2.2 Algorithmen (2), S. 21; INFWF 10-3.3.2 Algorithmen (2), S. 28;
BY	SEK I	RS- IT 2.7 Logik und Robotik, S.699; GYM 9/10 LPLUS INF - Modellieren, Implementieren, Anwenden, Softwareprojekte
BE	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
BB	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
HB	SEK II	GYM OS INF-Algorithmen und Datenstrukturen, S. 6; GYM OS INF-Imperative Programmierung, S. 7
HH	SEK I	Stadtteil 9/10 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 20; GYM 9 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 19
HE	SEK I	ohne curricularen Vorgaben
MV	SEK I	GYM 5 INF-3 Programmieren? Kinderleicht!, S.16; GYM 6 INF-3 Entscheidungen treffen und Spiele gestalten, S.19 GYM 7 INF-3 Spiele entwickeln, S.22; GYM 8 INF-3 Sensorgesteuerte Anwendungen entwickeln, S.25; GYM 9 INF-3 Problemlösen durch Programmieren, S.32
NI	SEK I	KC-INF LF Algorithmisches Problemlösen; S.19; KC-INF LF Automatisierte Prozesse, S.22; SEK 2 KC-INF LF1 Algorithmen und Datenstrukturen, S.14; SEK 2 KC-INF LF1 Informationen und Daten, S.16; ; SEK 2 KC-INF LF1 Automaten und Sprachen, S.19
NW	SEK I, II	RS 9/10 WPF TECHNIK 2.3 Inhaltsfeld 7: Kommunikations- und Digitaltechnik S.23; 5/6 KLP INF - Algorithmen, S. 17, 18; 5/6 KLP INF - Automaten und künstliche Intelligenz, S. 18; SEK 2 KLP GOS INF - 2 Algorithmen, S. 21 ff; KLP GOS INF - 3 Formale Sprachen und Automaten, S. 22
RP	SEK I	IPS 5 INF - Informatiksysteme und Netze, S. 7; IGS/GYM INF-2.1 Grundlagen der Informationsverarbeitung, S. 17; IGS/GYM INF-2.2 Algorithmisches Problemlösen, S. 20
SL	SEK I, II	GYM 9 INF - Imperative Programmierung, S. 3; INF - Algorithmik, S. 3; GYM OS INF GOS-Funktionsweise von Computersystemen, S.9ff.

SN	SEK I	GYM 7 INF LB 3: Computer verwenden – Komplexaufgabe, S. 7; GYM 8 INF LB 2: Daten verarbeiten, S.10
ST	SEK I, II	GYM 9 INF 3.2 Algorithmen interpretieren und entwickeln, S.15 ff.; GYM 11/12 INF 3.4 Kurs 3 Software Engineering und Projektarbeit, S. 23
SH	SEK I	INF PB1 Modellieren und Strukturieren, S. 12; INF PB2 Implementieren, Programmieren, Realisieren, S. 13; ; FA Physik, Variabilität S.13
TH	SEK I	GYM 10 INF - 2.3 Algorithmen, S. 14 ff.; GYM 10 INF 2.5.1 Technische Informatik, S. 18ff.

Anlagen
Aufgabe 3: Lichtautomatik

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.

Weiterführende Informationen

[1] Wikipedia: *Nebenläufigkeit*.

Name: _____ Klasse: ___ Datum: _____

Lösungsblatt Aufgabe 3

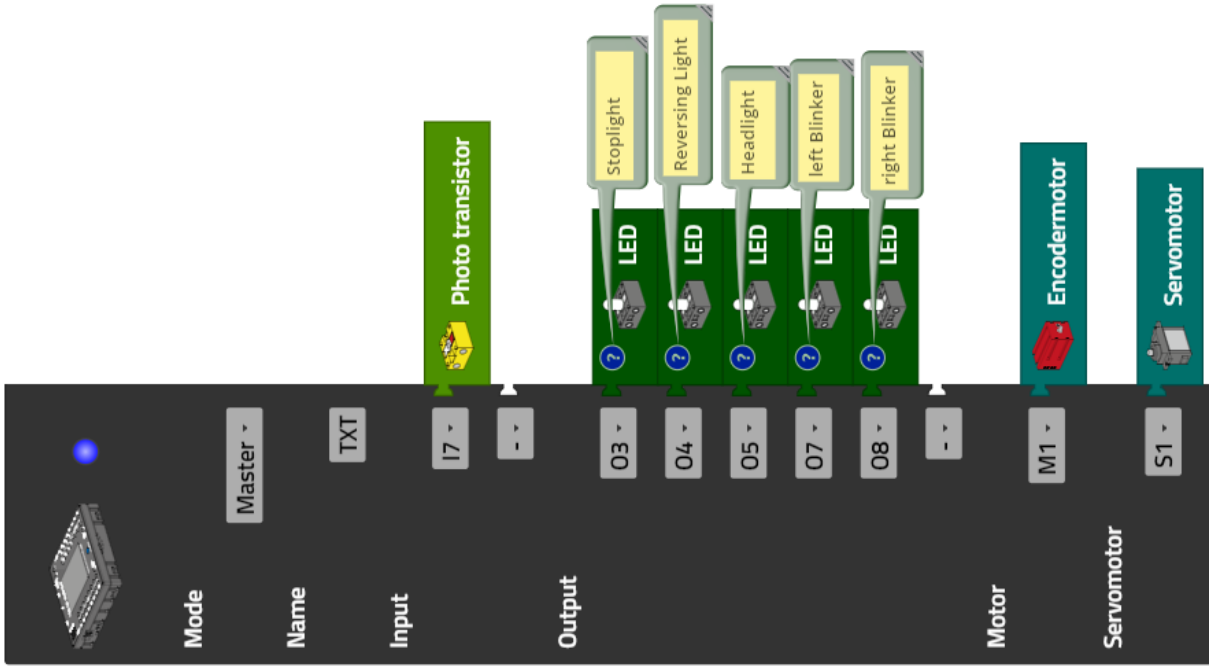
Lichtautomatik

Die Threads fragen jeweils einen „Trigger“ ab: den auslösenden Zustand wie die Geschwindigkeit bzw. die Drehrichtung des Motors oder den Einschlagwinkel des Servos. Eine gute Übung für den Nutzen von nebenläufigen Prozessen: Die Steuerung der Beleuchtung wird von der Motorsteuerung vollständig entkoppelt; dadurch wird das Programm übersichtlicher und ist weniger anfällig für Programmierfehler.

Konstruktionsaufgabe

In den folgenden Teilaufgaben werden lediglich die LED, der Fototransistor, der Motor und der Servo-Motor benötigt.

Konfiguration der Sensoren:



Programmieraufgaben

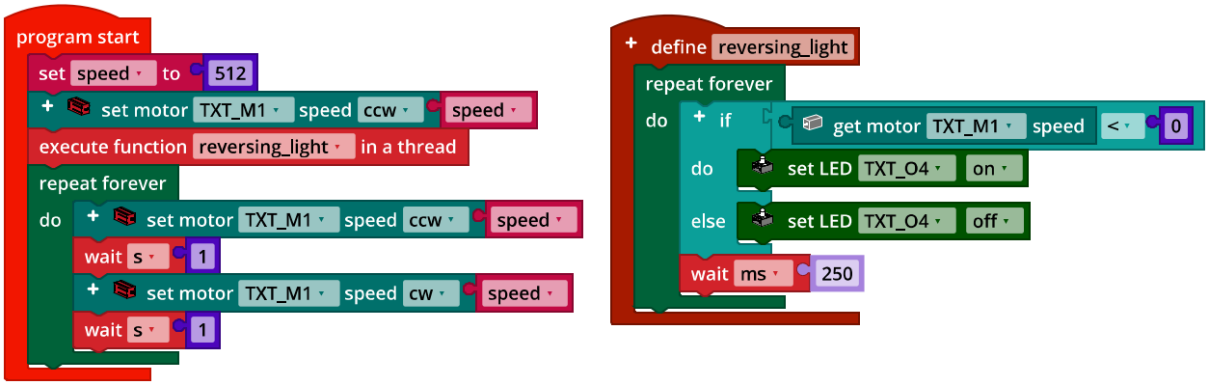
Der für die Aktivierung der LEDs jeweils maßgebliche Zustand wird direkt oder über Variablen (als Semaphore) im jeweiligen Thread abgefragt. Das Hauptprogramm enthält in den folgenden Programmbeispielen einfache Testroutinen, mit denen die Funktion der Lichtautomatik überprüft werden kann.

Das Programm „*Automatic_Ligthing.ft*“ enthält alle Threads für die Beleuchtung zusammengefasst.

1. Rückfahrlicht

Im Fall des Rückfahrlichts muss die Richtung der Motorumdrehung ausgewertet werden.

Programm (Beispiel) Rückfahrlicht mit Testprogramm:



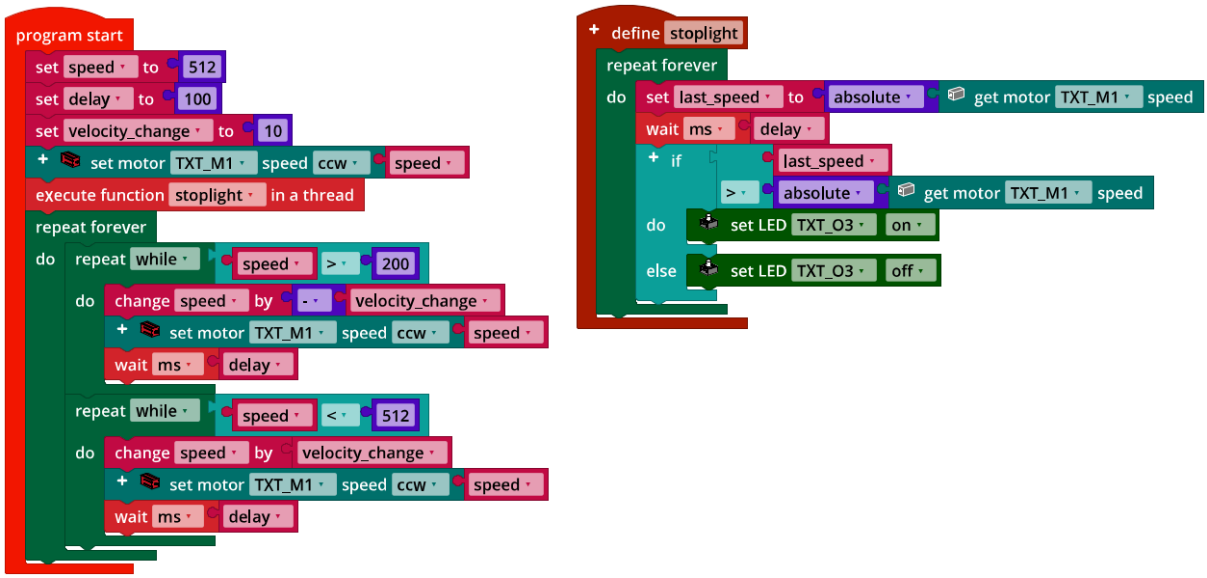
Reversing_Light.ft

2. Bremslicht

Damit das Bremslicht leuchten kann, wenn der Motor verzögert, ist die Messung aufeinanderfolgender Geschwindigkeiten des Motors erforderlich.

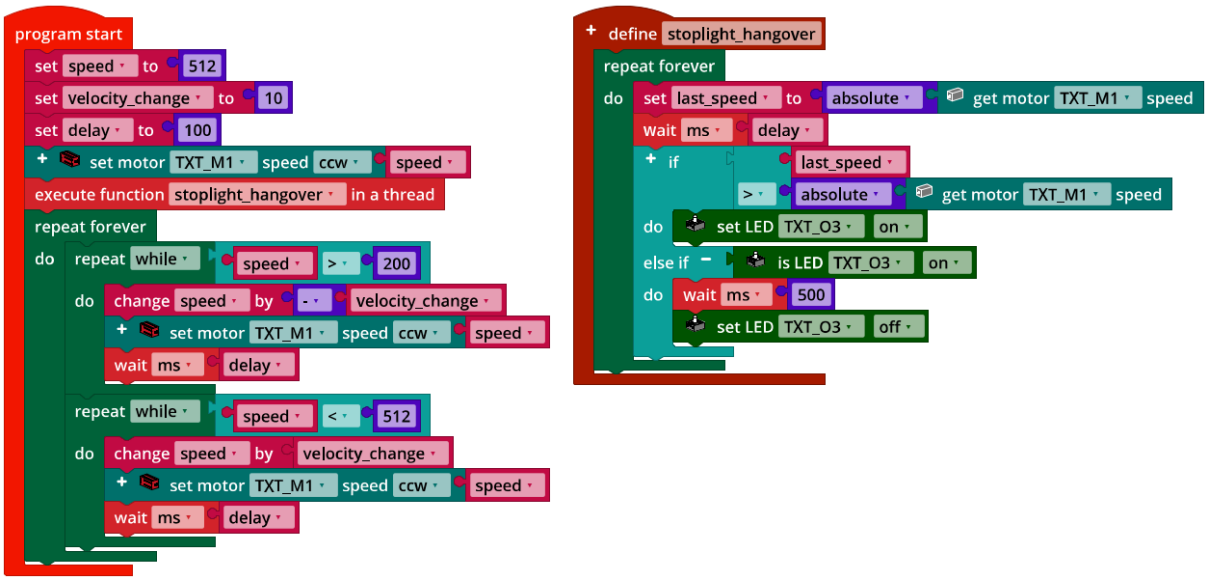
Zu beachten ist: Damit das Bremslicht auch bei einer Rückwärtsfahrt korrekt aufleuchtet, müssen die Beträge der Geschwindigkeiten verglichen werden, um eine Verzögerung festzustellen.

2a. Programm (Beispiel) Bremslicht mit Testprogramm:



Stoplight.ft

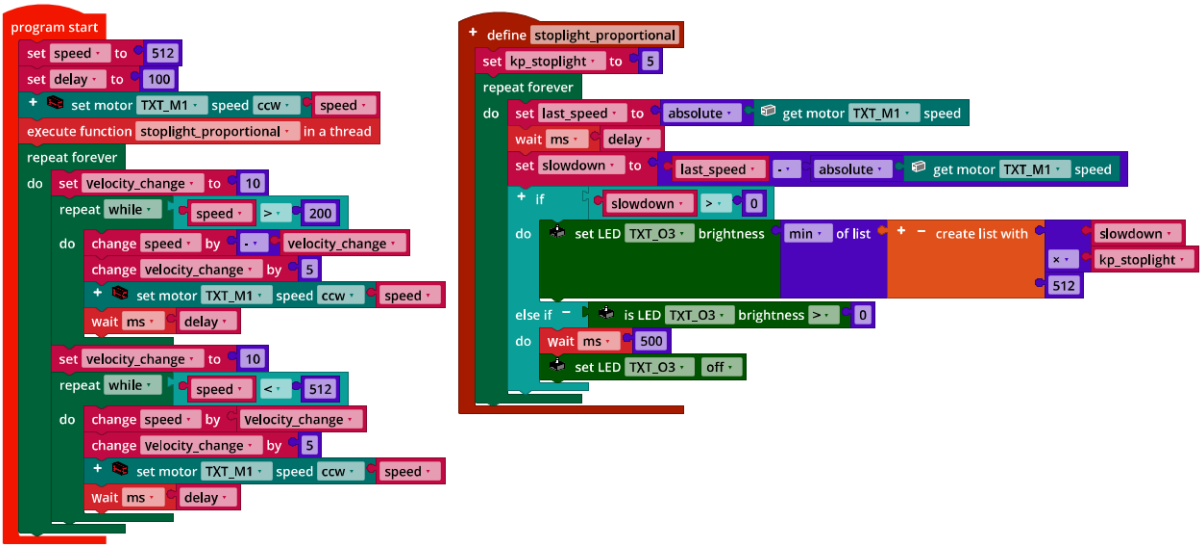
2b. Programm (Beispiel) Bremslicht mit Nachleuchten und Testprogramm:



Stoplight_hangover.ft

2c. Die Messdauer im Thread „stoplight“ hat erheblichen Einfluss auf die Größe der Geschwindigkeitsänderung; mit diesem Wert muss ein wenig experimentiert werden, um eine passende Reaktion bei einem realistischen Bremsverhalten abzubilden. Mit einem Proportionalitätsfaktor (k_p) kann der Einfluss der Geschwindigkeitsänderung auf die Helligkeit festgelegt werden.

Programm (Beispiel) Bremslicht mit proportionaler Helligkeit und Testprogramm:



Stoplight_proportional.ft

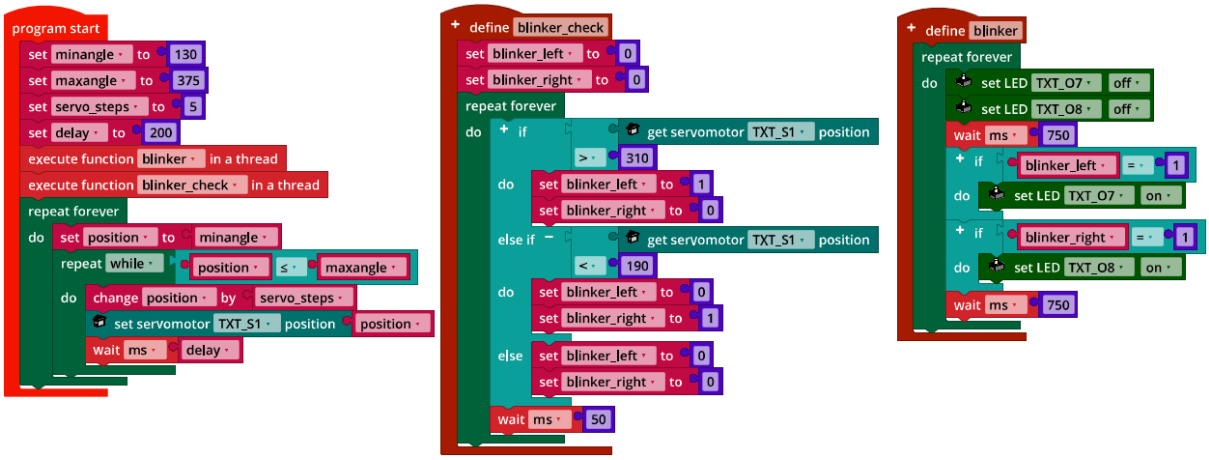
3. Blinker

Das Funktionsprinzip eines Blinkers ist bereits aus Aufgabe 3 des Robotics TXT 4.0 Base Set bekannt.

Für den Blinker werden mehrere Zustandsvariablen benötigt: eine je Blinker, die abhängig vom Lenkeinschlag (also der Position des Servo-Motors) angibt, ob der rechte oder der linke Blinker aktiviert ist, und ein Zustand, der unterscheidet, ob die Blinkleuchte (falls der Blinker aktiviert ist) ein- oder ausgeschaltet ist.

Das folgende Programmbeispiel löst die Aufgabe mit zwei Threads. Vorteil dieses Lösungsansatzes: Er kann sehr einfach um einen Warnblinker erweitert werden.

Programm (Beispiel) Blinker mit Testprogramm:



Blinker.ft

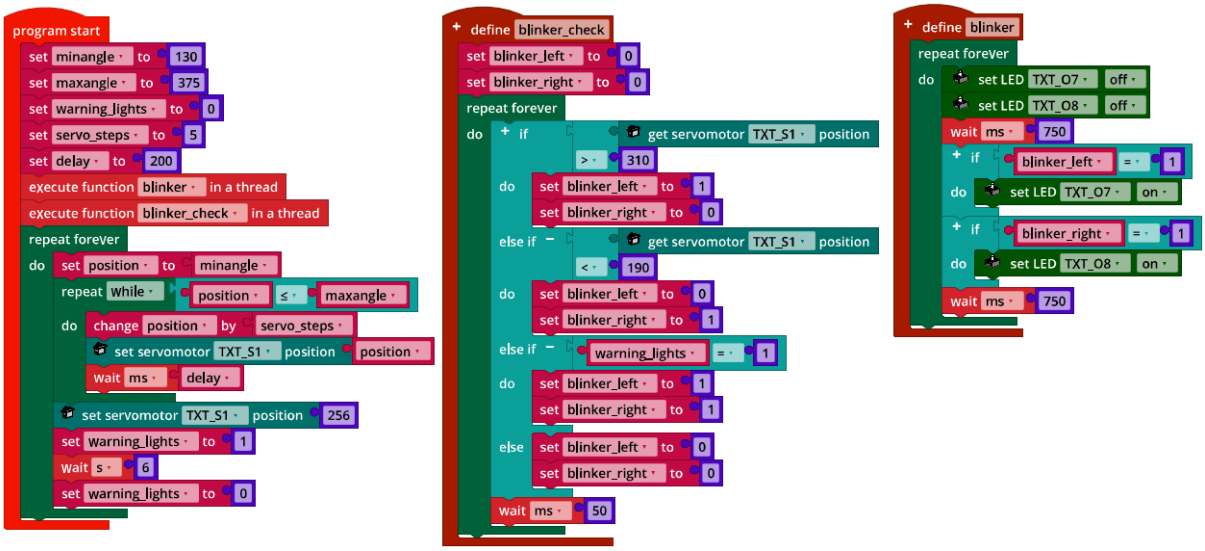
Die Status-Variablen „blinker_left“ und „blinker_right“ dienen als Semaphore zwischen dem Thread „blinker_check“ und „blinker“.

Experimentieraufgaben

1. Warnblinker

Der Status des Warnblinkers kann dem Thread über eine Variable als Semaphor signalisiert werden. Dann genügt es, den Blinker-Prüf-Thread zu erweitern: Wenn der Warnblinker eingeschaltet ist, werden beide Blinker aktiviert und im Blinker-Thread ein- und ausgeschaltet.

Programm (Beispiel) Warnblinker mit Testprogramm:

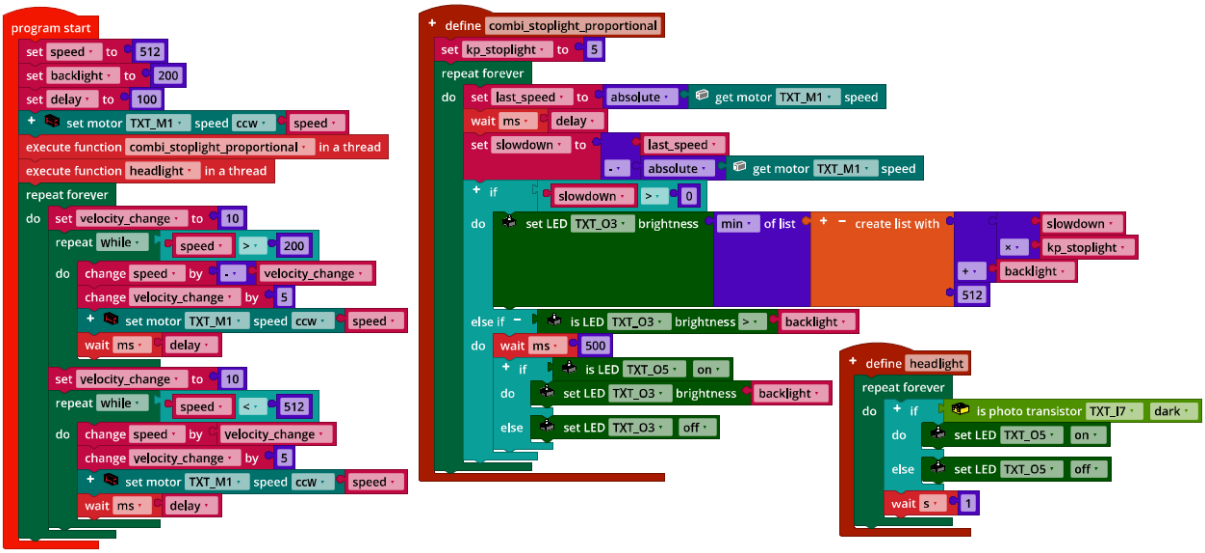


Blinker_with_Warning_Lights.ft

2. Abblendlicht

Die Helligkeit des Rücklichts darf nur geändert werden, wenn die LED nicht gleichzeitig als Bremslicht genutzt wird. Nutzt man im Thread für das Bremslicht den Status der Frontscheinwerfer als Indikator dafür, ob das Abblendlicht aktiviert ist, dann kann die Steuerung des kombinierten Rück- und Bremslichts in demselben Thread erfolgen.

Programm (Beispiel) Abblendlicht (Testprogramm):



Headlight.ft

Anlagen

Aufgabe 3: Lichtautomatik

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.

Weiterführende Informationen

[1] Wikipedia: *Nebenläufigkeit*.

Name: _____ Klasse: ___ Datum: _____

Aufgabe 4

Einparkhilfe

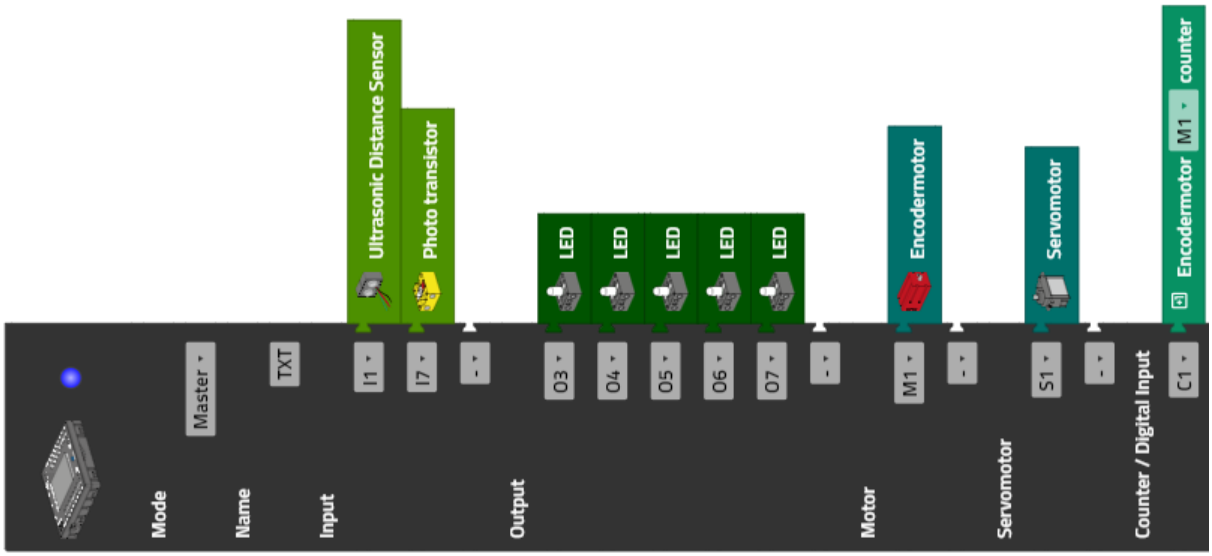
Wir statten das Fahrzeug nun mit einem weiteren wichtigen Fahrerassistenzsystem aus: einer Einparkhilfe. Damit soll es selbstständig eine ausreichend große Parklücke finden und in diese rückwärts in zwei Zügen einparken.

Konstruktionsaufgabe

Montiere den Ultraschallsensor so an deinem Fahrzeug, dass er den Abstand des Fahrzeugs zu Objekten am rechten Straßenrand messen kann.

Programmieraufgaben

Konfiguration der Sensoren:



1. Parklücke finden

Nun sollst du ein Programm entwickeln, das im Abstand von etwa 2 cm an der rechten Fahrbahnbegrenzung entlang steuert und dabei mit Hilfe des Ultraschallsensors eine ausreichend große Parklücke sucht. Die Parklücken kannst du durch Objekte (bspw. Kartons) begrenzen. Am Ende der ersten ausreichend großen Lücke soll das Fahrzeug an der Stelle halten, an der es mit dem Einparkvorgang beginnen kann.

Bestimme dazu zunächst manuell, wie groß (also wie lang und wie breit) eine Parklücke mindestens sein muss, damit dein Fahrzeug rückwärts einparken kann. Berücksichtige dabei die Streuung des Ultraschall-Signals: Wann erkennt der Sensor den Anfang und das Ende einer Parklücke, wenn diese durch Objekte begrenzt sind, die sich im Abstand von etwa 2 cm rechts neben der Fahrbahnbegrenzungslinie befinden?

Bestimme schließlich, wie weit das Fahrzeug hinter dem Ende der Parklücke zum Stehen kommen muss, damit es mit dem Einparkvorgang beginnen kann.

1a. Veranschauliche den Einparkvorgang und deine Messungen mit einer Zeichnung.

1b. Skizziere ein Zustandsübergangsdiagramm.

1c. Programmiere die Parklückensuche. Verwende dabei den Spurhalteassistenten aus Aufgabe 2.

Teste dein Programm auf dem Fahrbahnabschnitt des beiliegenden Bogens mit in unterschiedlichen Abständen aufgestellten Hindernissen rechts neben der Fahrbahnbegrenzung.

2. Einparkmanöver

Im zweiten Schritt soll das Fahrzeug, nachdem es eine Parklücke gefunden hat, rückwärts in zwei Zügen einparken.

2a. Bestimme zunächst wieder manuell (Interface Test), wie viele Impulse das Fahrzeug erst mit Lenkeinschlag nach rechts und anschließend mit Lenkeinschlag nach links zurücksetzen muss.

2b. Wie viele Impulse werden dann noch benötigt, damit das Fahrzeug in der Mitte (des zum Einparken benötigten Teils) der Parklücke zum Stehen kommt?

2c. Programmiere den Einparkvorgang und teste ihn auf dem Fahrbahnabschnitt.

Experimentieraufgaben

1. Einparkassistent

Erweitere deinen Einparkassistenten nun um die in Aufgabe 3 entwickelten Funktionen, damit er beim Einparken alle Anforderungen der Straßenverkehrsordnung erfüllt (Bremslicht, Rückfahrlicht, Blinker, Abblendlicht). Ergänze das Programm um ein „Erfolgssignal“, das nach Abschluss des Einparkvorgangs ertönt, und lass‘ dabei den Warnblinker fünf Mal aufleuchten.

2. Berechnung des Einparkvorgangs

Den Einparkvorgang kann man auch berechnen: Die Mitte des Differenzials beschreibt zwei Kreisabschnitte, deren Radien du aus dem Lenkeinschlagswinkel ableiten kannst.

2a. Zeichne die Phase des Einparkvorgangs, in der das Fahrzeug rückwärts fährt.

2b. Berechne aus den Winkeln und einigen Setzungen über die richtigen Abstände zur Seitenlinie die dafür benötigten Impulse.

Vergleiche das Ergebnis mit den von dir in der Programmieraufgabe experimentell bestimmten Werten.

Anlagen

Einparkhilfe

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Test-Fahrstrecke

Weiterführende Informationen

[1] VDA: *Automatisierung: Von Fahrerassistenzsystemen zum automatisierten Fahren*. September 2015.

[2] Nina Tetzlaff: *Autonome Straßenfahrzeuge*. Erfinderaktivitäten 2018/2019, DPMA, S. 62-81.

[3] VW: *Park Assist Steering 2.0*. Design and function. Service Training, 2021.

Aufgabe 4: Einparkhilfe

Das Fahrzeug wird mit einer Einparkhilfe ausgestattet, mit deren Hilfe es eine ausreichend große Parklücke suchen und in zwei Zügen rückwärts einparken soll.

Thema

Programmierung eines Fahrerassistenzsystems (Einparkhilfe).

Lernziel

- Ausmessung eines Objekts aus der Bewegung unter Berücksichtigung der Streuung des Ultraschallsensors
- „Teile und herrsche“: Vorgehensweise zur Komplexitätsreduktion durch Zerlegung eines Problems in voneinander unabhängige Teilprobleme
- Trigonometrische Berechnung eines Bewegungsablaufs

Zeitaufwand

Am autonomen Fahrzeug ist lediglich der Ultraschallsensor seitlich zu befestigen.

Für die Entwicklung der Programme zur Lösung der Aufgaben benötigen Schülerinnen und Schüler zwei bis vier Unterrichtsstunden (90-180 Minuten).

Bezug Curriculum

Land	Stufe/Fächer	Bezüge
BW	SEK I	GYM 8/9/10 NWT-3.2.4.3 Steuerungsabläufe (Ampelsteuerung, Robotik) (7), Informationsverarbeitung - Autonomes Fahren (8), S.27; IMP 8-3.1.1.2 Algorithmen (1), S. 28ff; INFWF 8-3.1.2 Algorithmen (1), S. 15; INFWF 9-3.2.2 Algorithmen (2), S. 21; INFWF 10-3.3.2 Algorithmen (2), S. 28;
BY	SEK I	RS- IT 2.7 Logik und Robotik, S.699; GYM 9/10 LPLUS INF - Modellieren, Implementieren, Anwenden, Softwareprojekte
BE	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
BB	SEK I	7-10 Informatik-3.8 Projektmanagement und 3.9 Physical Computing (Wahlthemenfeld), S. 27
HB	SEK II	GYM OS INF-Algorithmen und Datenstrukturen, S. 6; GYM OS INF-Imperative Programmierung, S. 7

HH	SEK I	Stadtteil 9/10 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 20; GYM 9 INFORMATIK-M2 Prozesse analysieren und modellieren, S. 19
HE	SEK I	ohne curricularen Vorgaben
MV	SEK I	GYM 5 INF-3 Programmieren? Kinderleicht!, S.16; GYM 6 INF-3 Entscheidungen treffen und Spiele gestalten, S.19 GYM 7 INF-3 Spiele entwickeln, S.22; GYM 8 INF-3 Sensorgesteuerte Anwendungen entwickeln, S.25; GYM 9 INF-3 Problemlösen durch Programmieren, S.32
NI	SEK I	KC-INF LF Algorithmisches Problemlösen; S.19; KC-INF LF Automatisierte Prozesse, S.22; SEK 2 KC-INF LF1 Algorithmen und Datenstrukturen, S.14; SEK 2 KC-INF LF1 Informationen und Daten, S.16; ; SEK 2 KC-INF LF1 Automaten und Sprachen, S.19
NW	SEK I, II	RS 9/10 WPF TECHNIK 2.3 Inhaltsfeld 7: Kommunikations- und Digitaltechnik S.23; 5/6 KLP INF - Algorithmen, S. 17, 18; 5/6 KLP INF - Automaten und künstliche Intelligenz, S. 18; SEK 2 KLP GOS INF - 2 Algorithmen, S. 21 ff; KLP GOS INF - 3 Formale Sprachen und Automaten, S. 22
RP	SEK I	IPS 5 INF - Informatiksysteme und Netze, S. 7; IGS/GYM INF-2.1 Grundlagen der Informationsverarbeitung, S. 17; IGS/GYM INF-2.2 Algorithmisches Problemlösen, S. 20
SL	SEK I, II	GYM 9 INF - Imperative Programmierung, S. 3; INF - Algorithmik, S. 3; GYM OS INF GOS-Funktionsweise von Computersystemen, S.9ff.
SN	SEK I	GYM 7 INF LB 3: Computer verwenden – Komplexaufgabe, S. 7; GYM 8 INF LB 2: Daten verarbeiten, S.10
ST	SEK I, II	GYM 9 INF 3.2 Algorithmen interpretieren und entwickeln, S.15 ff.; GYM 11/12 INF 3.4 Kurs 3 Software Engineering und Projektarbeit, S. 23
SH	SEK I	INF PB1 Modellieren und Strukturieren, S. 12; INF PB2 Implementieren, Programmieren, Realisieren, S. 13; ; FA Physik, Variabilität S.13
TH	SEK I	GYM 10 INF - 2.3 Algorithmen, S. 14 ff.; GYM 10 INF 2.5.1 Technische Informatik, S. 18ff.

Anlagen
Einparkhilfe

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Test-Fahrstrecke

Weiterführende Informationen

[1] VDA: *Automatisierung: Von Fahrerassistenzsystemen zum automatisierten Fahren*. September 2015.

[2] Nina Tetzlaff: *Autonome Straßenfahrzeuge*. Erfinderaktivitäten 2018/2019, DPMA, S. 62-81.

[3] VW: *Park Assist Steering 2.0*. Design and function. Service Training, 2021.

Name: _____ Klasse: ___ Datum: _____

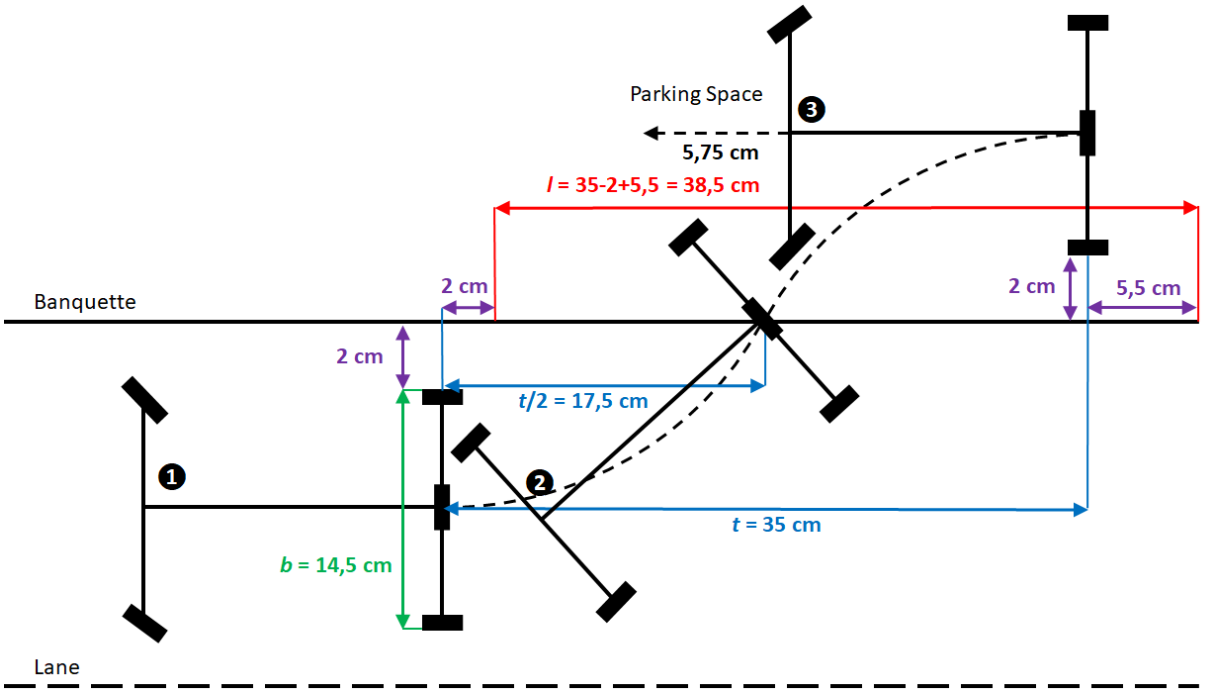
Lösungsblatt Aufgabe 4

Einparkhilfe

Programmieraufgaben

1. Parklücke finden

Der Einparkvorgang lässt sich mit demontiertem Servo-Hebel leicht per Hand nachvollziehen. Die Länge der erforderlichen Parklücke kann am Einfachsten mit Hilfe entsprechender Bleistift-Markierungen auf der Fahrbahn ausgemessen werden (siehe Abbildung).



Parallel_Parking_Model.jpg

Breite der Parklücke: Die Parklücke muss mindestens so breit sein wie das Fahrzeug, also b = 14,5 cm, zuzüglich 2 cm bis zur Fahrbahnbegrenzung.

Die Lücke sollte zusätzliche 1,5 cm breiter sein, da die Reifen beim Lenkeinschlag über die Fahrzeugbreite hinausragen.

Ist der Ultraschallsensor so montiert, dass er ab dem äußeren Rand des Fahrzeugs misst, sind weitere 2 cm bis zur Seitenlinie sowie deren Breite (2 cm) hinzuzuaddieren.

Also befindet sich rechts neben dem Fahrzeug eine ausreichend breite Lücke, wenn der Sensor **mindestens 22 cm** bis zum nächsten Objekt misst.

Länge der Parklücke: Beim Einparkvorgang in zwei Zügen muss sich der Mittelpunkt des Differentials zunächst etwa 17,5 cm mit maximalem Lenkeinschlag nach rechts rückwärts bewegen, bevor er die Mitte der Seitenlinie kreuzt, anschließend weitere 17,5 cm mit maximalem Lenkeinschlag nach links. Zu dieser Strecke von 35 cm sind die Länge des Fahrzeughecks (5,5 cm, gemessen von der Mitte des Differenzials bis zum äußersten Ende des Fahrzeugs) zu addieren und der „Start-Abstand“ von 2 cm (zwischen der Position des Differenzials beim Start des Einparkvorgangs und der Stelle, an der das linke Vorderrad den Seitenstreifen kreuzt) zu subtrahieren. Damit summiert sich die benötigte Mindestlänge der Parklücke auf

$$l = 38{,}5\ \text{cm}$$

Aus der Mindestlänge der Parklücke l lässt sich die Zahl der für diese Distanz erforderlichen Impulse i nach der folgenden Formel bestimmen (siehe Aufgabe 1):

$$i = l \cdot \frac{1}{20{,}5} \cdot \frac{13}{7} \cdot 63{,}9\ \frac{1}{\text{cm}} \approx l \cdot\ 5{,}789\ \frac{1}{\text{cm}}$$

Berücksichtigen wir den angepassten Umrechnungsfaktor aus dem Test in Aufgabe 1 so korrigiert sich die Umrechnung zu

$$i = l \cdot \frac{100}{0{,}0017} \approx l \cdot\ 5{,}882\ \frac{1}{\text{cm}}$$

Damit erfordert die gesuchte Parklückenlänge $l = 38{,}5\ \text{cm}$ also $\boldsymbol{i \approx 227}$ **Impulse**.

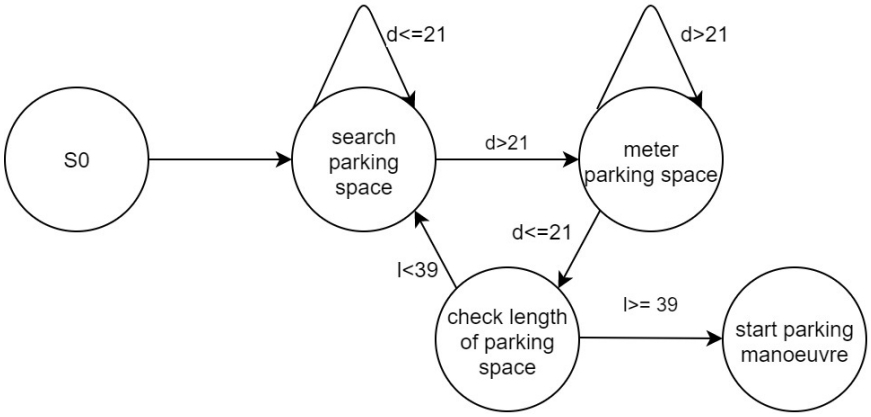
Streuung des Ultraschall-Signals: Wenn Hindernisse im Abstand von 4 cm rechts neben der Fahrbahnbegrenzung stehen, beträgt der Abstand zum Ultraschallsensor 8 cm. Durch den Streuwinkel des Ultraschallsensors wird die Parklücke erst erkannt, wenn die Mitte des Ultraschallsensors bereits 3 cm in die Parklücke „hineinragt“. Das Ende der Parklücke wird entsprechend bereits 3 cm vor der Stelle erkannt, an der die Sensormitte den äußeren Rand des Hindernisses erreicht. Misst der Sensor also über eine Strecke von 32,5 cm (oder $\boldsymbol{i' \approx 191}$ **Impulse**) eine Breite von mindestens 22 cm, ist die gefundene Parklücke lang genug.

Für die Messung der Parklückenlänge ist es irrelevant, an welcher Stelle der Ultraschallsensor seitlich am Fahrzeug angebracht ist. Für den Start des Einparkvorgangs muss das Fahrzeug so zum Stehen kommen, dass sich die Hinterachse (bzw. das Differential) genau 2 cm hinter dem Ende der Parklücke (also dem äußersten Rand des Hindernisses) befindet. Dazu muss es, nachdem der Ultraschallsensor das Ende der Parklücke erkannt hat,

- weitere 3 cm (um die das Ende der Parklücke zu früh erkannt wird)
- plus obige 2 cm
- zuzüglich des Abstands von der Mitte des Ultraschallsensors zur Hinterachse

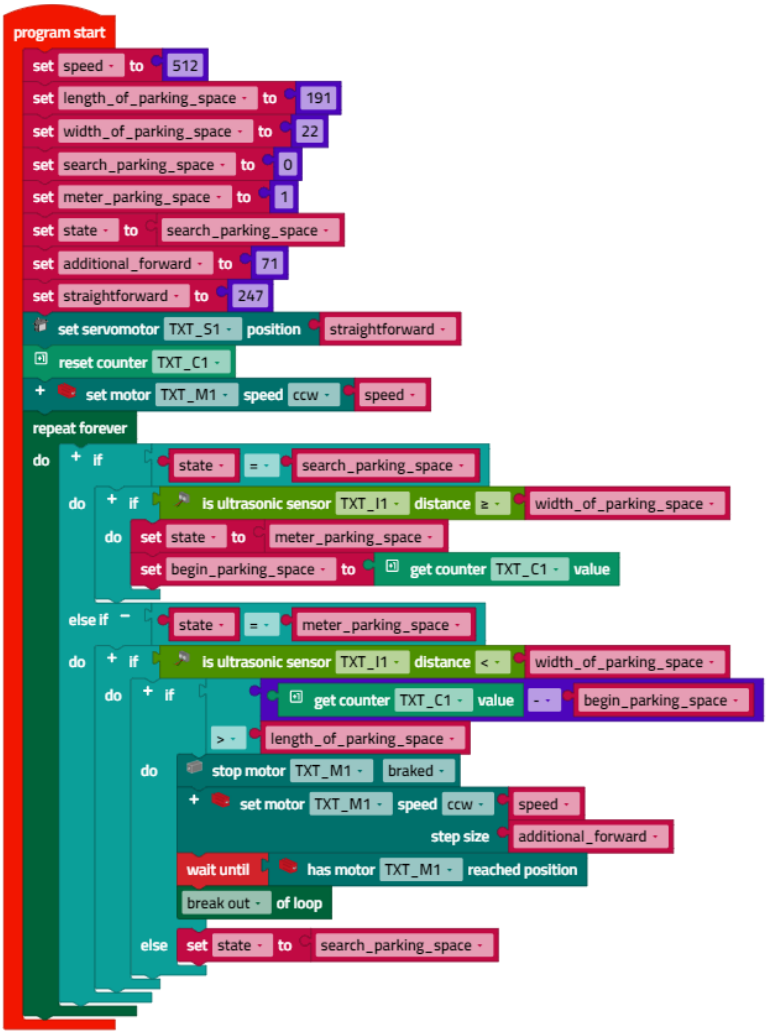
weiterfahren. Ist der Ultraschallsensor beispielsweise 7 cm vor der Hinterachse montiert, muss das Fahrzeug nach Erkennen des Parklückenendes weitere 12 cm (oder **71 Impulse**) fahren.

1b. Zustandsübergangsdiagramm:



State-Transistion_Diagram_Find_Parking_Space.drawio

1c. Programm (Beispiel) Parklückensuche:



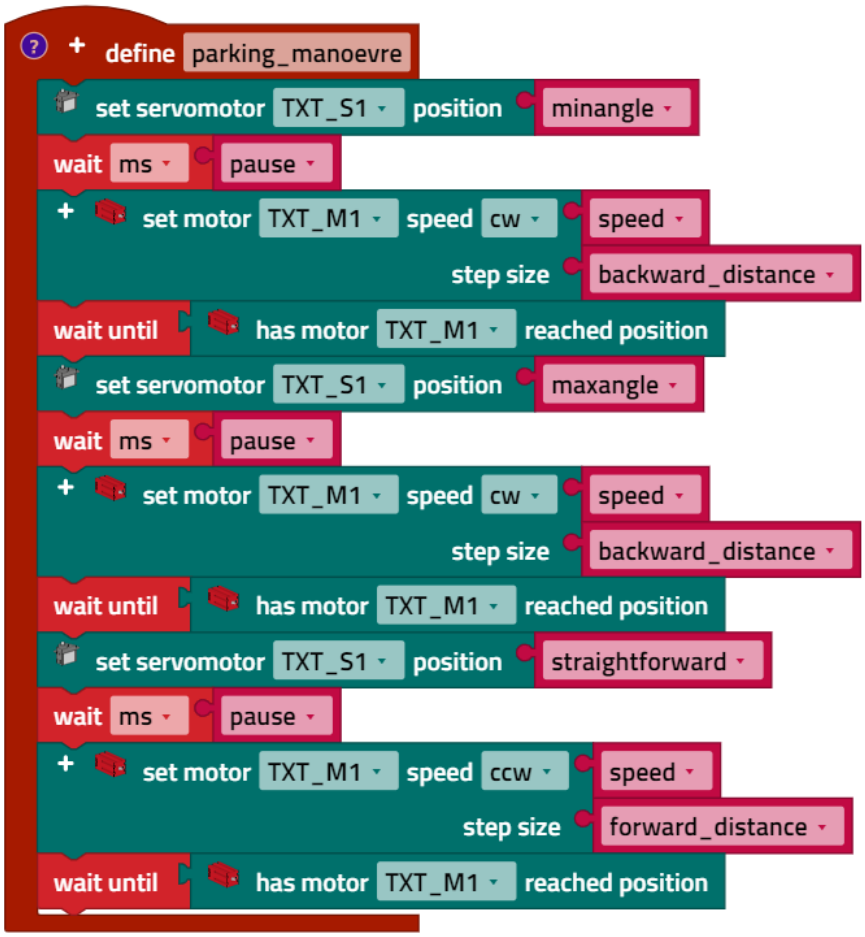
Find_Parking_Space.ft

2. Einparkmanöver

2a. Die Bestimmung der zurückzulegenden Impulse gelingt am Einfachsten mit dem Interface-Test: Das Fahrzeug wird manuell eingeparkt; dabei werden die vom Encoder gelieferten Impulse im Interface-Test abgelesen und notiert. Für das Zurücksetzen sind – mit nach rechts bzw. nach links eingeschlagener Lenkung – **jeweils 125 Impulse** erforderlich.

2b. Damit das Fahrzeug zum Schluss in der Mitte des benötigten Einparkbereichs der Parklücke steht, muss es nach dem Zurücksetzen 5,75 cm vorfahren; das entspricht etwa **34 Impulsen**.

2c. Programm (Beispiel) Einparkmanöver:



Parking_Manoevre.ft

Die Variablen „backward_distance“ und „forward_distance“ enthalten die zu fahrenden Impulse (125 bzw. 34). „minangle“ und „maxangle“ sind die maximal möglichen Einschlagwinkel des Servos im Servo-Halter (siehe Aufgabe 2); sie liegen etwa bei 130 bzw. 375. Für die kurze Pause nach einem Lenkeinschlag („pause“) sind 250 ms eine gute Wahl.

Experimentieraufgaben

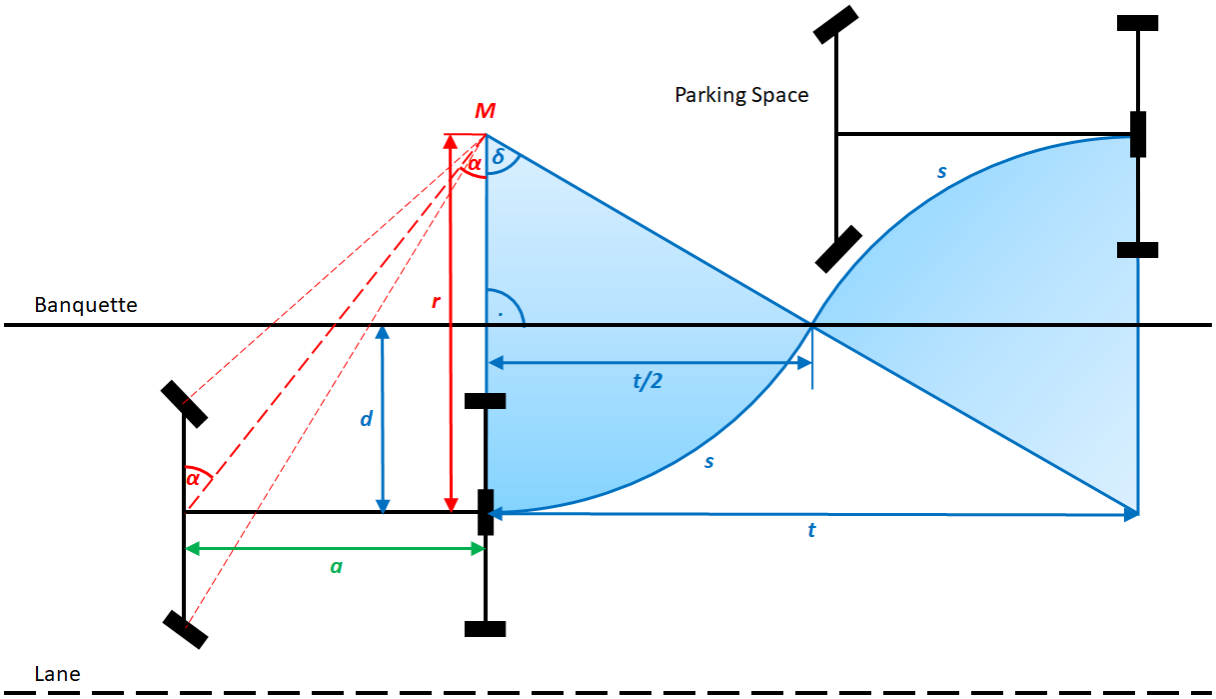
1. Einparkassistent

Eine Lösung der Aufgabe ist im Programm *Autonomous_Parking.ft* zusammengefasst.

Der Blinker darf nicht automatisch aktiviert werden, da sonst mitten im Einparkvorgang auf den linken Blinker gewechselt wird.

2. Berechnung des Einparkvorgangs

Der Mittelpunkt des Differentials bewegt sich beim Einparkvorgang exakt entlang zweier Kreissegmente der Länge s. Der Mittelpunkt M des ersten Kreises entspricht bei unserer Achsschenkellenkung dem Mittelpunkt des Wendekreises: Die Verlängerungen der beiden vorderen Achsstummel der Lenkung und der Hinterradachse treffen sich exakt in diesem Punkt.



Mathematical_Model_Parallel_Parking.jpg

Wir berechnen nun die Länge dieses Kreissegments s. Damit wissen wir, wie viele Encoder-Impulse wir benötigen, um dieses Segment abzufahren. Es gilt: s verhält sich zum Kreisumfang ($= 2\pi \cdot r$) wie δ zu 360°, also:

$$s = \frac{\pi \cdot \delta \cdot r}{180°}$$

Den Winkel δ können wir durch den Radius r ausdrücken, denn in dem rechtwinkligen Dreieck, das M und die beiden Schnittpunkte der Kreissegmentlinien mit der Seitenlinie bilden, gilt:

$$\cos\delta = \frac{(r-d)}{r}$$

Für den Abstand d vom Achsmittelpunkt zur (Mitte der) Seitenlinie nehmen wir an, dass sich das Fahrzeug etwa 2 cm links von der Fahrbahnbegrenzung befindet. Dann gilt bei einer Fahrzeugbreite von b = 14,5 cm und mit unserer 2 cm breiten Seitenlinie:

$$d = 10{,}25\ \text{cm}$$

Den Radius r des Wendekreises berechnen wir aus dem Dreieck, das von M und den beiden Mittelpunkten der Vorder- und Hinterachse des Fahrzeugs gebildet wird. Den Achsabstand bezeichnen wir mit a:

$$\tan\alpha = \frac{a}{r}\text{, also: } r = \frac{a}{\tan\alpha}$$

Der Winkel α entspricht dabei dem (maximalen) Einschlagswinkel unseres Servo-Hebels. Bei unserem Fahrzeug liegt der Einschlagswinkel der Lenkung bei jeweils etwa 38°. Den Achsabstand können wir messen: $a = 15{,}5\ \text{cm}$.

Wir erhalten schließlich (mit $\alpha = 38°$):

$$s = \frac{\pi}{180°}\cdot\arccos\left(\frac{a - d\cdot\tan\alpha}{a}\right)\cdot\frac{a}{\tan\alpha} \approx 21{,}15\ \text{cm}$$

Das entspricht etwa 122 Impulsen.

Jetzt können wir auch die benötigte Länge der Parklücke berechnen. Nach dem *Satz des Pythagoras* gilt:

$$r^2 = \left(\frac{t}{2}\right)^2 + (r-d)^2$$

Damit folgt (mit $\alpha = 38°$):

$$t = 2\cdot\sqrt{\left(\frac{a}{\tan\alpha}\right)^2 - \left(\frac{a}{\tan\alpha} - d\right)^2} \approx 34{,}73\ \text{cm}$$

Zu diesem Wert müssen wir noch die Länge des Hecks des Fahrzeugs hinzurechnen (5,5 cm) und 2 cm abziehen, die wir über die Parklücke hinaus fahren. Unsere Parklücke sollte also mindestens 38,23 cm lang sein - das entspricht ziemlich genau dem experimentell bestimmten Wert.

Anlagen

Einparkhilfe

Erforderliches Material

- PC für Programmentwicklung, lokal oder über Web-Schnittstelle.
- USB-Kabel oder BLE- bzw. WLAN-Verbindung für die Übertragung des Programms auf den TXT4.0.
- Test-Fahrstrecke

Weiterführende Informationen

[1] VDA: *Automatisierung: Von Fahrerassistenzsystemen zum automatisierten Fahren*. September 2015.

[2] Nina Tetzlaff: *Autonome Straßenfahrzeuge*. Erfinderaktivitäten 2018/2019, DPMA, S. 62-81.

[3] VW: *Park Assist Steering 2.0*. Design and function. Service Training, 2021.