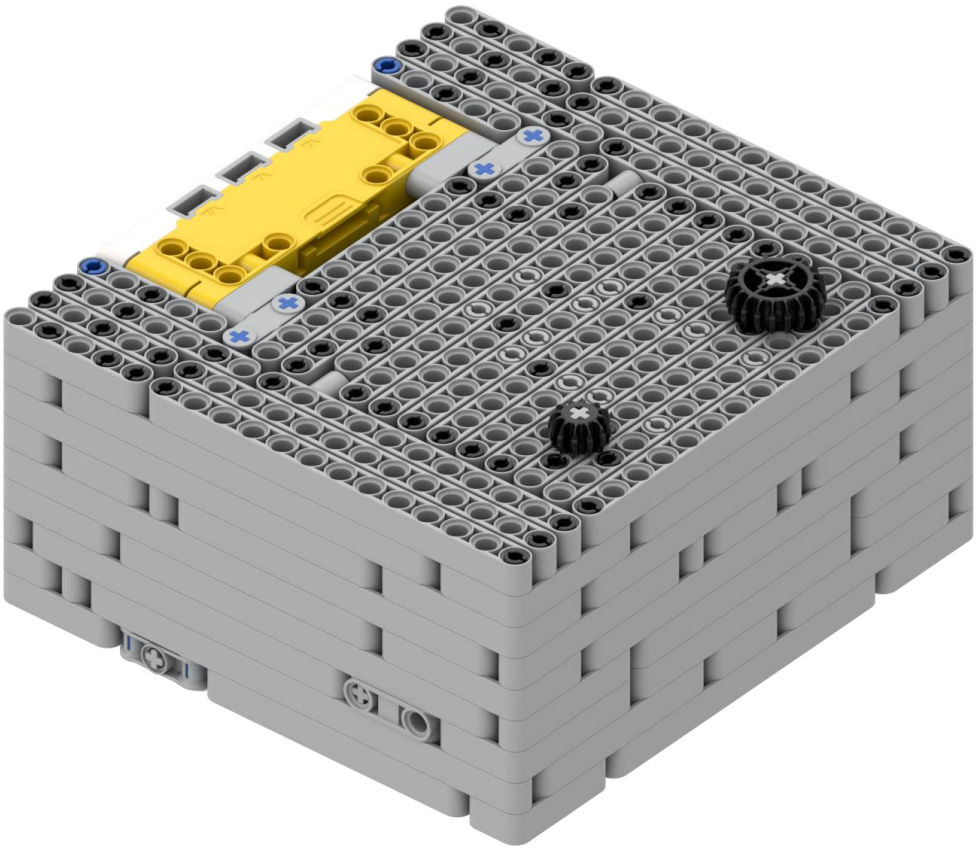
GYMNASIUM OTTOBRUNN

Robot Design

Dokumentation des Roboters und seiner Programmierung



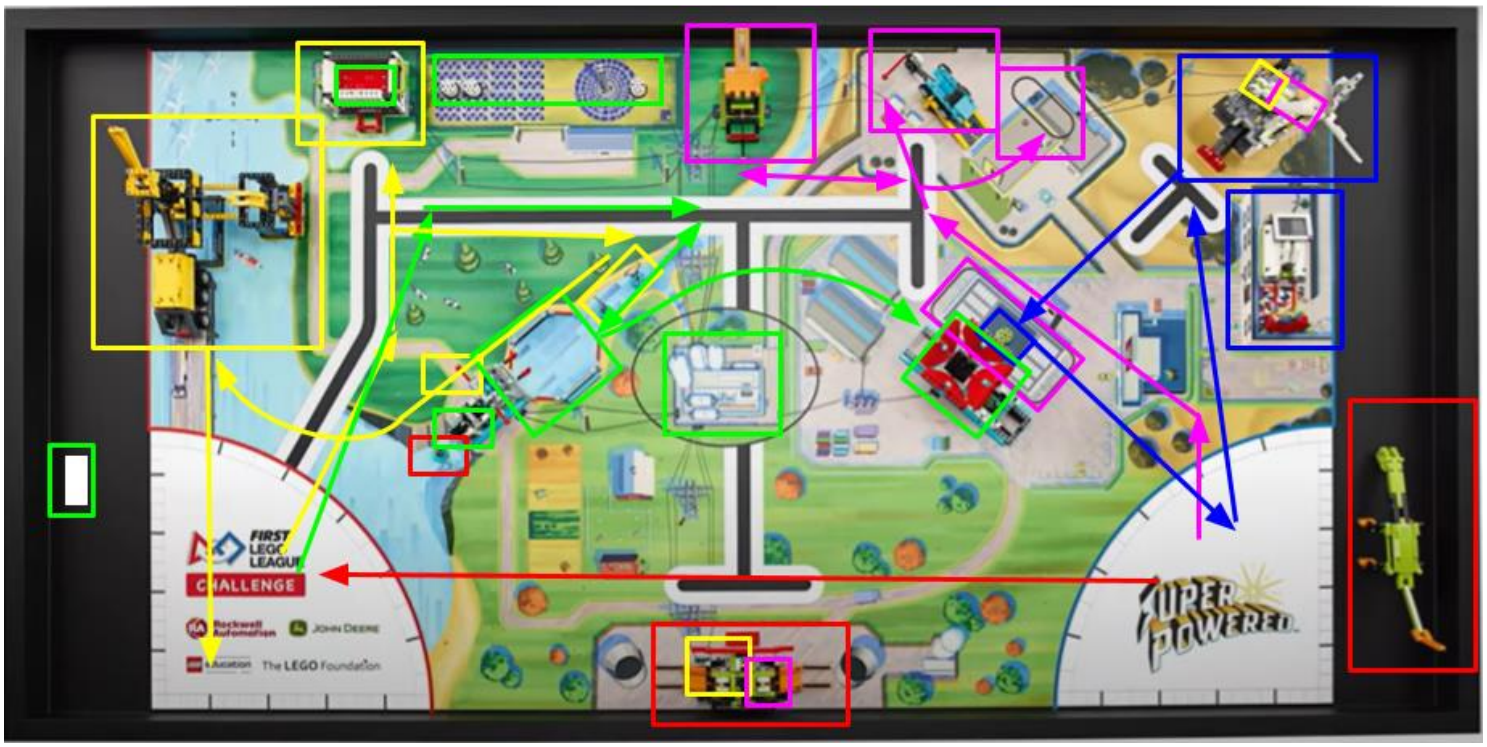
GYMNASIUM OTTOBRUNN

Inhalt

Strategie

Rundenplanung

- Ziel der Planung: möglichst viele Punkte in möglichst wenig Zeit möglichst zuverlässig erreichen
- Mit unserer Planung erreichbare Punktzahl: 410 (maximale Punktzahl)
- Aufgaben in einer Runde gruppieren → weniger Zeit in der Startzone
- Leicht Wechselbare Aufsätze → weniger Zeit in der Startzone
- Möglichst gleichmäßige Verteilung der Punkte auf die Runden → reduziertes Risiko



Entwicklung der Planung

- Erstellen einer Lohnenswertigkeitstabelle (Siehe nächste Seite)
- Planung mit Bau und Programmierung
- Zu Beginn: Auswahl einiger Aufgaben, danach Feststellung, dass alle Aufgaben in Zeit machbar sind

Ziel: häufiges und genaues Ausrichten an Missionsmodellen und Linien, fehlertoleranter Bau und Programmierung

Lohnenswertigkeit

- Erstellung zu Beginn der Saison
- Faktoren: Distanz, Schwierigkeit, Punkte
- Berechnung mit Formel $Lohnenswertigkeit = \frac{Punkte}{Distanz + Schwierigkeit}$
- Beginn mit einfachen Runden

Aufgabe	Variation	Punkte	Distanz	Schwierigkeit	Lohnenswertigkeit
A08 Watch Television	Ganz	20	2	2	5.00
A11 Hydroelectric Dam	Ganz	20	2	2	5.00
A03 Energy Storage	Ganz	35	4	5	3.89
A07 Wind Turbine	Ganz	30	4	4	3.75
A15 Rechargable Battery	Ganz (3)	15	3	1	3.75
A14 Toy Factory	Ganz	25	3	4	3.57
A03 Energy Storage	Teils(3 drinnen)	30	4	5	3.33
A14 Toy Factory	Teils(2 and dino)	20	3	4	2.86
A03 Energy Storage	Teils(2 drinnen + tray)	25	4	5	2.78
A02 Oil Platform	Ganz	25	2	8	2.50
A04 Solar Farm	Ganz	20	7	1	2.50
A05 Smart Grid	Mit bonus	30	10	2	2.50
A07 Wind Turbine	Teils(2 units raus)	20	4	4	2.50
A08 Watch Television	Teils	10	2	2	2.50
A15 Rechargable Battery	Teils(2)	10	3	1	2.50

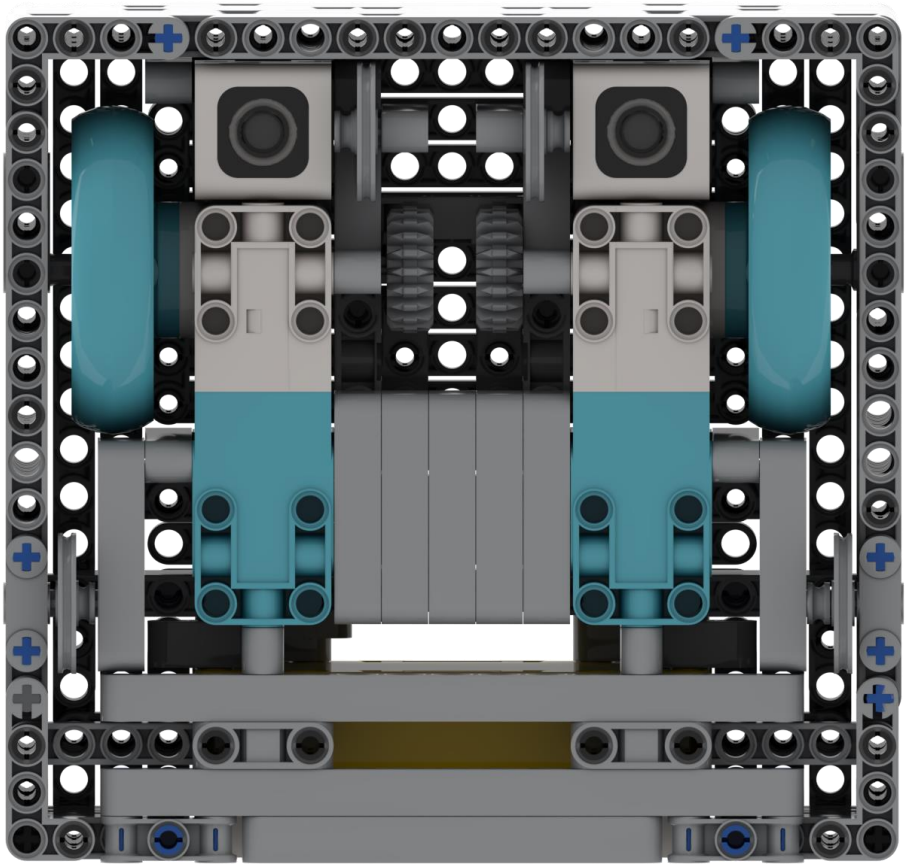
A09 Dinosaur Toy	Ganz	30	10	3	2.31
A01 Innovation Project Model	Ganz	20	8	1	2.22
A03 Energy Storage	Teils(2 drinnen)	20	4	5	2.22
A14 Toy Factory	Teils(1 and dino)	15	3	4	2.14
A14 Toy Factory	teils(3 no dino)	15	3	4	2.14
A02 Oil Platform	Teils(2 teile drinnen + truck)	20	2	8	2.00
A10 Power Plant	Ganz	25	4	9	1.92
A03 Energy Storage	Teils(1 drinnen + tray)	15	4	5	1.67
A05 Smart Grid	nur wir	20	10	2	1.67
A09 Dinosaur Toy	Teils(Batterie, aber nicht links)	20	10	3	1.54
A09 Dinosaur Toy	Teils(Energy packet und links)	20	10	3	1.54
A02 Oil Platform	Teils(3t eile drinnen)	15	2	8	1.50
A02 Oil Platform	Teils(1 drinnen + truck)	15	2	8	1.50
A13 Power-to-X	Ganz(max 3)	15	8	2	1.50
A14 Toy Factory	Teils(2 no dino)	10	3	4	1.43
A04 Solar Farm	Teils(2 raus vom Kreis)	10	7	1	1.25
A07 Wind Turbine	Teils 1 unit raus)	10	4	4	1.25
A15 Rechargable Battery	Teils(1)	5	3	1	1.25
A12 Water Resevoir	Ganz	20	7	10	1.18
A01 Innovation Project Model	Teils	10	8	1	1.11
A03 Energy Storage	Teils(1 drinnen)	10	4	5	1.11
A06 Hybrid Car	Ganz	20	8	10	1.11
A02 Oil Platform	Teils(2 teile drinnen)	10	2	8	1.00
A13 Power-to-X	teils(2)	10	8	2	1.00
A12 Water Resevoir	Teils(1 in and 1 hook)	15	7	10	0.88
A09 Dinosaur Toy	Teils(Energy Paket)	10	10	3	0.77

A10 Power Plant	Teils(2 energy units)	10	4	9	0.77
A14 Toy Factory	Teils(1 no dino)	5	3	4	0.71
A04 Solar Farm	Teils(1 raus vom Kreis)	5	7	1	0.63
A12 Water Resevoir	teils(only 2 in)	10	7	10	0.59
A12 Water Resevoir	teils(only hook)	10	7	10	0.59
A06 Hybrid Car	teils(nur ein von den bedingungen)	10	8	10	0.56
A02 Oil Platform	Teils(1 drinnen)	5	2	8	0.50
A13 Power-to-X	Teils(1)	5	8	2	0.50
A10 Power Plant	Teils 1 energy unit)	5	4	9	0.38
A12 Water Resevoir	Teils(only one in)	5	7	10	0.29

Gelb hinterlegt = wird gemacht

Bau

Baseroboter



- 2 große Motoren zum Fahren, 2 mittlere Motoren als Aktionsmotoren, 2 Bodenlichtsensoren zur Erkennung von Linien
- Entwicklung mit Bricklink Studio 2.0
- Hohe Stabilität
- Gelochte Oberseite zur Befestigung von Aufsätzen und Zahnrädern
- Unterschiedlich große Dockzahnräder für unterschiedliche Geschwindigkeits- und Kraftanforderungen
- Möglichkeit mit einem Dockzahnrad am Baseroboter vier Zahnräder an Aufsatz anzutreiben
- Positionierung der Aufsätze über den Rädern für verbesserte Gewichtsverteilung
- Untersätze möglich
- Kabelmanagement
- Ziel: Lichtsensoren so nah wie möglich an Antriebsrädern, kompakt, stabil, vielseitig

Verwendete Sensoren

Bodenlichtsensoren

- Folgen von Linien
- Ausrichten an Linien
- Fahren bis Linie
- Drehen bis Linie
- Tief und von Außenlicht abgeschirmt → erhöhte Genauigkeit
- Sehr nah an Rädern → schnelleres Folgen von Linien möglich

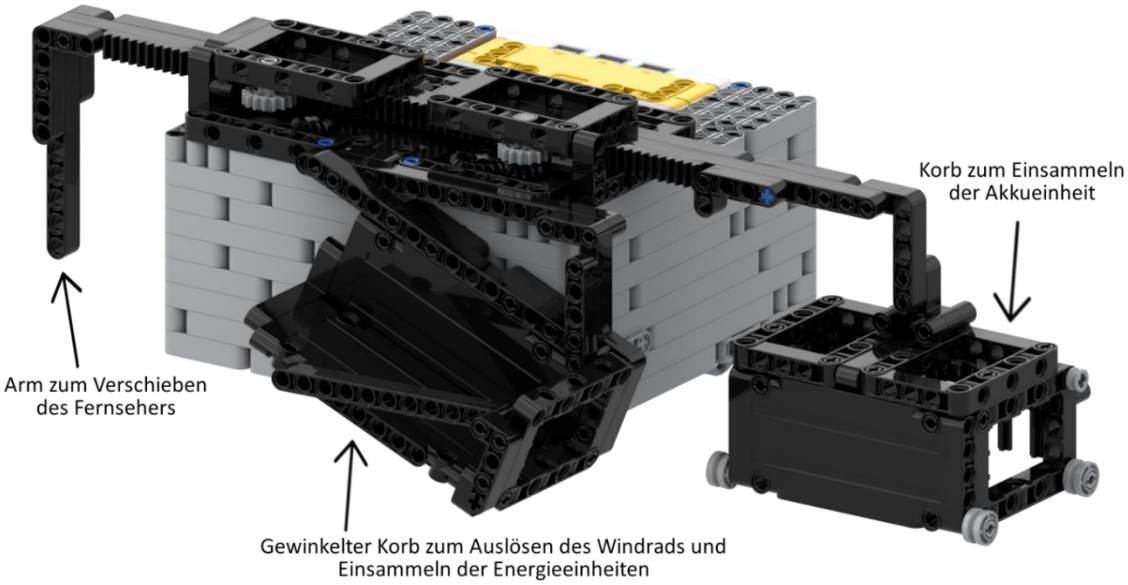
Gyrosensor

- Mit Gyrosensor gerade fahren
- Mit Gyrosensor drehen
- Mit Gyrosensor Kurven fahren
- Normalerweise: Bei Werten >|180| kommt es zur Signalumkehr, Angabe von Drehungen mehr als 180 nicht möglich
- →eigene Methode, die Signalumkehr erkennt und den echten Wert weiterzählt, sodass Drehungen >180° möglich werden

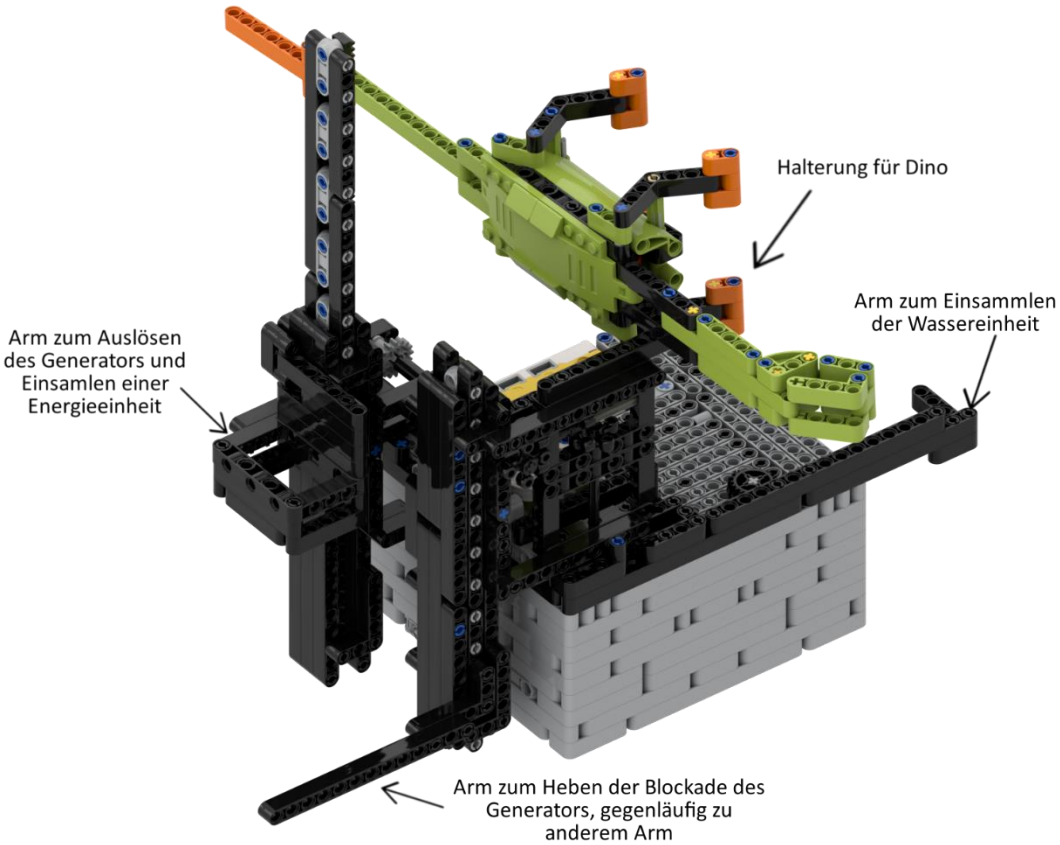
Räder

- Mittelgroß → schnell, aber genau
- hart → genau, viel Traktion, wenig Variablen aufgrund von Kompression
- Doppelt gelagert: Erhöhte Stabilität und Genauigkeit

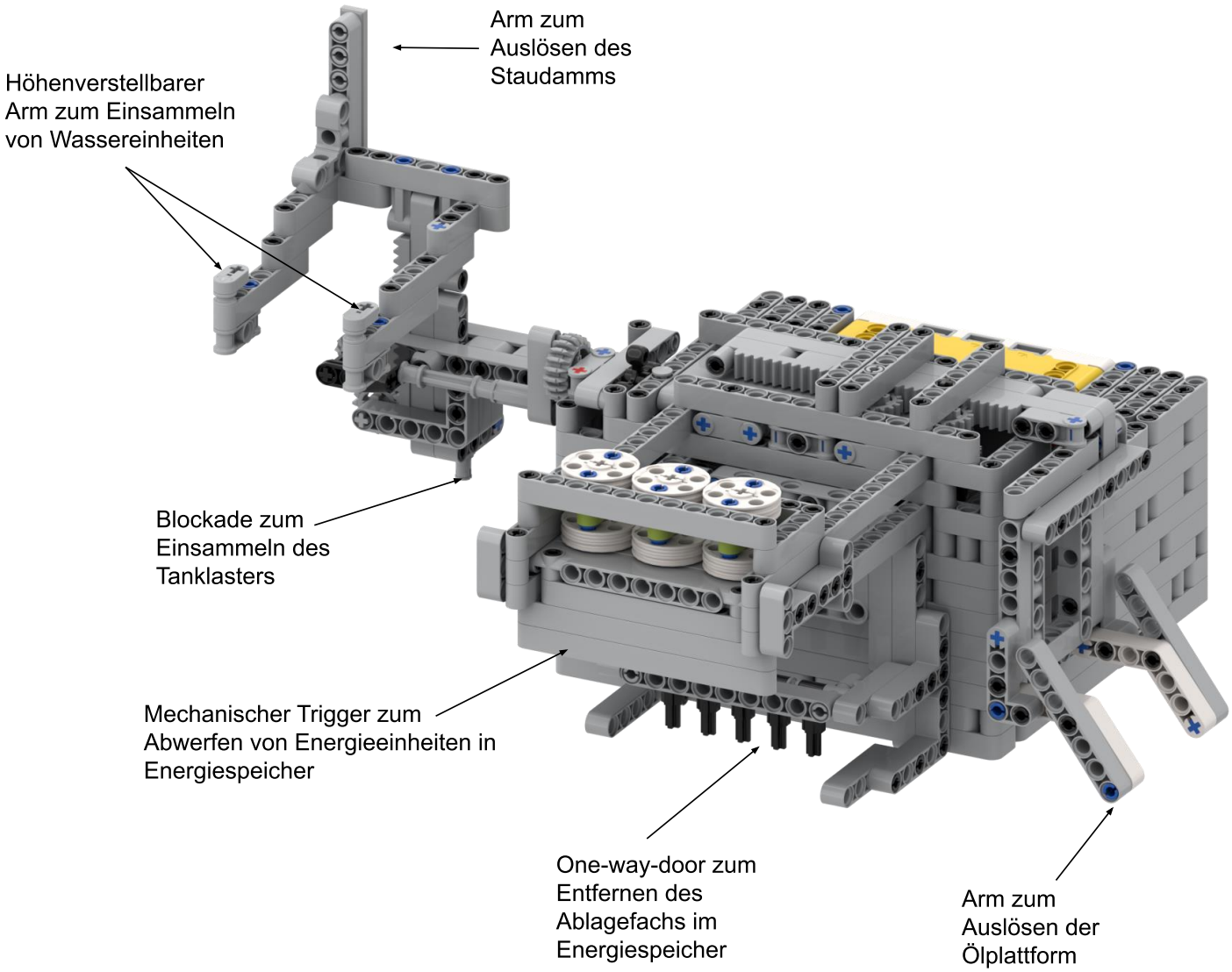
Runde 1 (50 Punkte, 15 Sekunden)



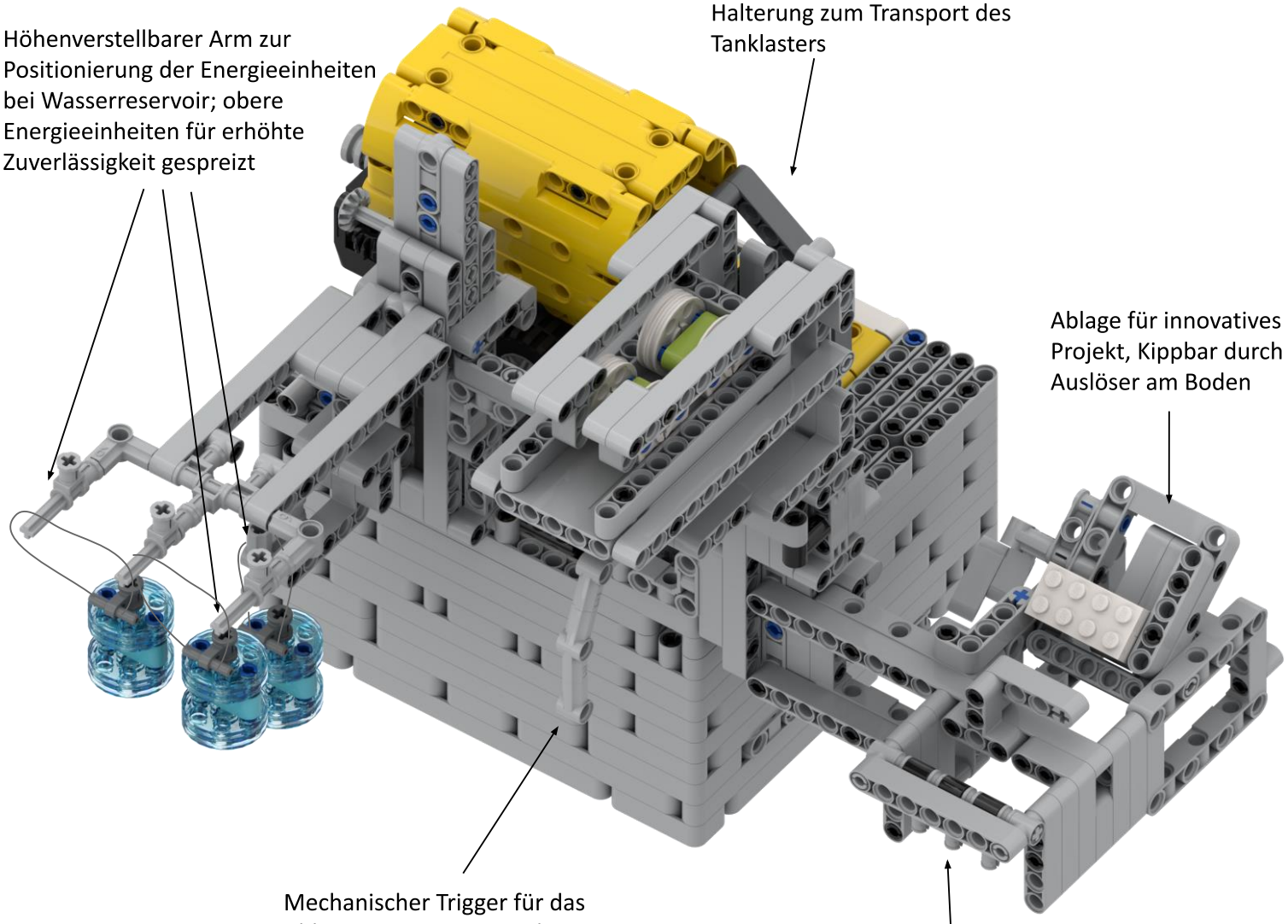
Runde 2 (55 Punkte, 9 Sekunden)



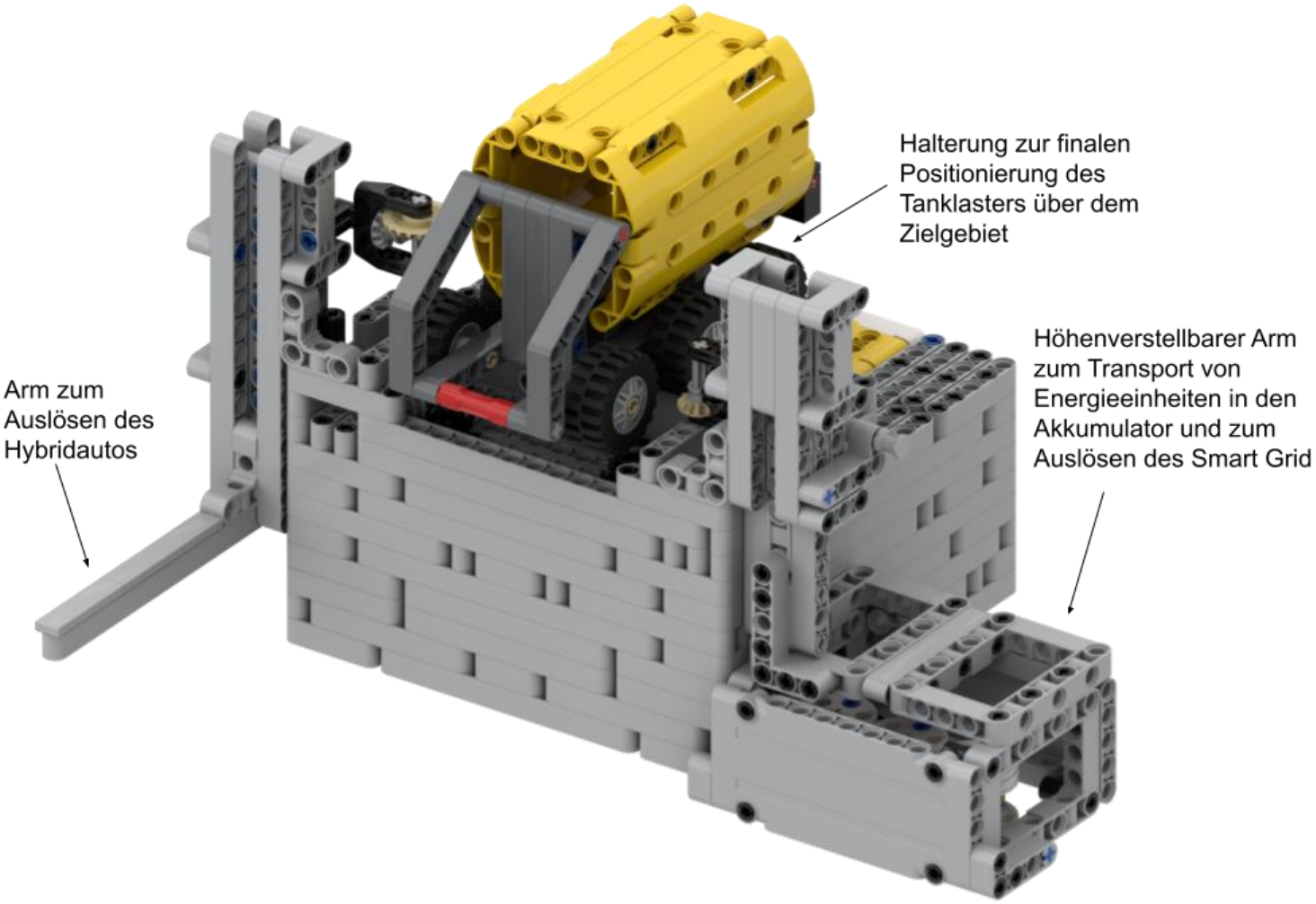
Runde 3 (80 Punkte, 26 Sekunden)



Runde 4 (90 Punkte, 25 Sekunden)



Runde 5 (65 Punkte, 22 Sekunden)



Programmierung

Effizienz

- Strukturierung unseres Codes in Klassen und Methoden für bessere Übersichtlichkeit
- Eigene Methoden mit sinnvollen Parametern für schnelles Programmieren
- Keine Nutzung von print Funktionen → mehr Schleifendurchläufe
- Verschiedene Varianten für unterschiedliche Methoden des Programms → kürzere Programme, weniger Redundanz
- Leichte Wartbarkeit, da kein Redundanter Code
- Ausführliche Dokumentation des Codes, für leichtere Reflexion und erhöhte Verständlichkeit für andere und das eigene Team (→ siehe Öffentliches Github Repository)
- Verwendung von sinnvollen Default Parametern, die sonst oftmals gleich sind → zeiteffizientere und übersichtliche Programmierung

Programmierumgebung

- Programmierung in Micropython
- Nutzung von Visual Studio Code für Github Support und Autocomplete
- Nutzung von Github für Versionsmanagement zum kollaborativem Arbeiten
  → Erhöhte Effizienz durch paralleles Arbeiten möglich

Main

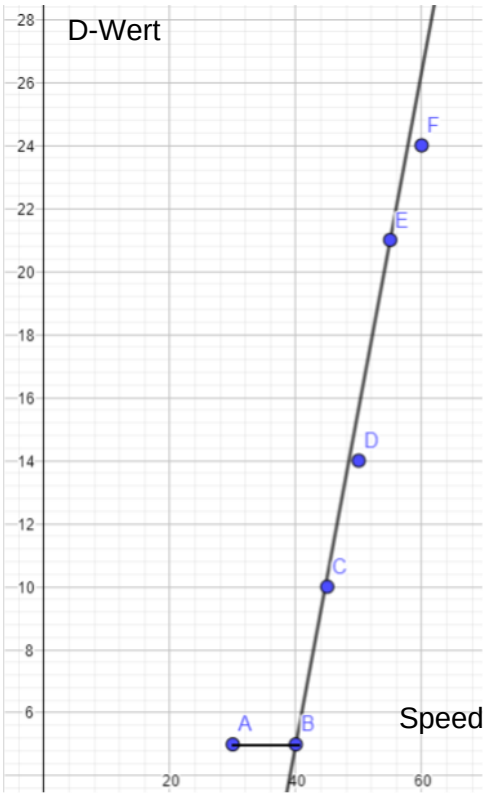
- Preloading von Programmen für erhöhte Effizienz während Runs
- Auswahl aller Runden möglich
- Einfache Unterscheidung der Methoden durch Anzeige der Rundennummer und einfärbung des Hauptknopfs
- Start der Runde per Knopfdruck

Abbruchmethode

- Abbruch der aktuell laufenden Runde ohne Abbruch der Main
- Vorteil: Zeitersparnis, da Runden schon vorgeladen sind, die aktuelle Runde ausgewählt bleibt, und das Programm sich nicht vollständig schließen muss

PID

- Alle Fahrmethoden nutzen PID - Regler
- P - Regler: Korrektur von Abweichungen
- I - Regler: Korrektur von konstanten Abweichungen wie Drifts, z.B. aufgrund von Gewichtsverteilung
- D - Regler: schnelle Reaktion auf plötzliche Änderungen
- Durch PID können längere Strecken ohne Ausrichten bewältigt werden
- Durch PID können lange Strecken in kürzerer Zeit bewältigt werden, da wir schnell und trotzdem genau fahren können
- Wegen variabler Geschwindigkeit: dynamisches Anpassen aller PID-Werte an aktuelle Geschwindigkeit für erhöhte Genauigkeit und Zuverlässigkeit



Verhalten des D-Werts in Abhängigkeit zur Geschwindigkeit

Eigene Programme

Gyrogerade

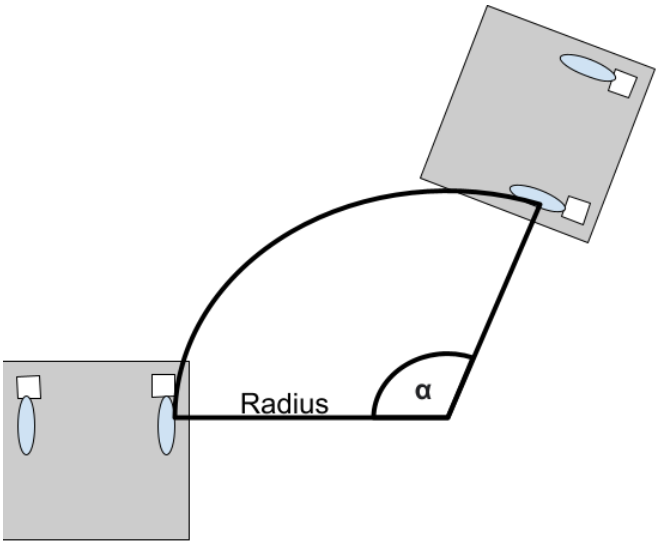
- Für gerades Fahren
- Ausgleichen der entstehenden Ungenauigkeiten, zum Beispiel durch Motoren, Gewichtsverteilung, Hindernisse etc. mit Gyrosensor und PID-Korrektursystem
- Beschleunigung und Verlangsamung für ruhiges und genaues Fahren (→ siehe Geschwindigkeitsberechnung)
- PID-Regler auf jede Geschwindigkeit optimal und einzeln angepasst
- Endbedingungen: → theoretisch alle Stop Methoden möglich
- Parallele Nutzung von Aktionsmotoren möglich (→ siehe Motorparallelisierung)

Gyrodrehung

- Für genaues Drehen
- Beschleunigung und Verlangsamung für ruhiges und genaues Fahren
- (→ siehe Geschwindigkeitsberechnung)
- Gyrosensor Kalibrierung im Programm
- Unterschiedliche Varianten für Drehen auf der Stelle/Fahren von Kurven
- Endbedingungen: → theoretisch alle Stop Methoden möglich

Arc-Rotation

- Fahren von Kurven mit einem gegebenen Radius und Winkel
- Erlaubt das Fahren von Kurven mit größerem Radius
- Beschleunigung und Verlangsamung für ruhiges und genaues Fahren
- (→ siehe Geschwindigkeitsberechnung)
- Endbedingungen: → theoretisch alle Stop Methoden möglich



Linienfolger

- Für exaktes Fahren und Folgen von Linien
- Beschleunigung und Verlangsamung für ruhiges und genaues Fahren
- (→ siehe Geschwindigkeitsberechnung)
- PID-Regler auf jede Geschwindigkeit einzeln angepasst
- Unterstützt fahren bis zu schwarzer/weißer Linie
- Lichtsensor kann individuell ausgewählt werden
- Seite der Linie auf der gefahren wird kann eingestellt werden
- Endbedingungen: Distanz, Linienerkennung (Überspringen von Linien möglich)

Methoden

- Verschiedene selbst programmierte Abbruchbedingungen einmal definieren und in verschiedenen Fahrmethoden nutzen
- Dadurch verringerte Code-Redundanz

stopLine (Lichtsensor, Lichtwert, erkennungsStart)

- stoppt, wenn der aktuelle Lichtwert unter dem angegebenen Lichtwert liegt
- Distanz ab bei der die Linienerkennung beginnt, kann selbst eingestellt werden (Überspringen von Linien möglich)
- Das Lichtlevel kann selbst angegeben werden

stopAlign (Lichtwert, Geschwindigkeit)

- stoppt an einer Linie
- Richtet sich dann an der Linie aus
- Das Lichtlevel kann selbst angegeben werden
- Die Geschwindigkeit für das Fahren mit der einen Seite muss auch übergeben werden

stopTangens (Lichtwert, Geschwindigkeit)

- stoppt an einer Linie
- Richtet sich an der Linie mithilfe des Tanges Verfahrens aus
- Das Lichtlevel kann selbst angegeben werden
- Die Geschwindigkeit kann selbst angegeben werden

stopTime (Zeit)

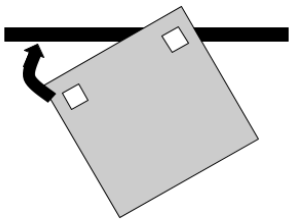
- stoppt, wenn eine bestimmte Zeit vergangen ist
- Vorteilhaft, wenn man sich z.B. an einer Wand ausrichten möchte

stopResistance (Widerstand)

- stoppt, wenn ein gewisser Widerstandswert überschritten wird
- Die Stärke des Widerstandes kann eingestellt werden

Linienausrichten

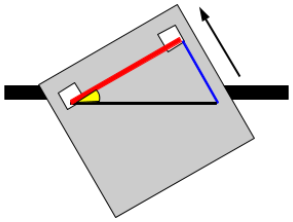
- Ausrichten an Linien für genaueres Fahren
- Kann sowohl an weißen als auch an schwarzen Linien genutzt werden
- Variante der Gyrogerade



Pfeil: Strecke, die noch gedreht werden muss, damit auch der zweite Lichtsensor unter dem Lichtwert liegt

Tangens - Linienausrichten

- Messung der Distanz zwischen Auftritt des ersten und zweiten Lichtsensors auf Linie
- Berechnung des Winkels um den gedreht werden muss mithilfe des Tangens
- Drehen des Winkels mit Gyrodrehung
- Vorteil: Unterschiedliche Latenzen während des Programms werden ausgeglichen



Blau: gefahrene Strecke
Rot: Abstand der Lichtsensoren
Gelb: Winkel, der mit Gyrodrehung ausgeglichen wird

Methode zur Geschwindigkeitsberechnung

- Dauer der Be- und Entschleunigung kann relativ zur gefahrenen Strecke angegeben werden
- Enthält allgemeine Formel zur Berechnung der Geschwindigkeit
- Berechnung der linearen Beschleunigung und Verlangsamung basierend auf Startgeschwindigkeit, Maximalgeschwindigkeit und Endgeschwindigkeit in Abhängigkeit zur bereits gefahrenen Distanz

Motorparallelisierung

- Keine native Unterstützung in Micropython von Legosseite
- Erlaubt gleichzeitige Benutzung der Fahr- und Aktionsmotoren
- Nutzung der yield Funktion in Python:
    - Umwandlung der Methode zum Bewegen des Aktionsmotors in Schritte in Listeneinträge
    - Bei jedem Schleifendurchlauf wird ein Schritt ausgeführt

Öffentliches Github Repository

- Aktuelle eigene Programme für andere zugänglich
- Erstes Suchergebnis für „FLL Code Python“, Zweites für „Spike Prime Python Code“
- Feedback und Verbesserungsvorschläge von anderen Teams
- Ausführliche Dokumentation mit Tutorials und Anwendungsbeispielen
- https://github.com/GO-Robot-FLL/Python-for-Spike-Prime

Testprozess

- Ausgedehnter Testprozess über 2 Monate
- Anfangs: Testen des Baus durch manuelles Bewegen des Roboters
- Danach: Grundlegende Programmierung der Runde, bei Bedarf Veränderung der Aufsätze
- Nach ausgiebigen Zuverlässigkeitstest: Beschleunigung der Runde
- Letzte Wochen vor dem Wettbewerb: Testen von vollständigen Runs inklusive Aufsatzwechsel, Optimierung von Programmierung und Üben der Aufsatzwechsel

Verbesserungen gegenüber letzter Saison

- Stärke der Beschleunigung und Verlangsamung abhängig von der gefahrenen Distanz und dem Geschwindigkeitsunterschied
- Neue Methoden wie arcRotation und die Stop Methoden
- Verringerte Code Redundanz
- Neue, eigene Abbruch Methode für schnelleres Abbrechen während Run und direktem Rückkehr in die Main
- Diverse Bugfixes
- Stabilisierung der Oberseite des Baseroboters
- Verwendung neuer Zahnradübersetzung zwischen Baseroboter und Aufsatz für erhöhte Flexibilität beim Bau
- Nutzung pythagoreischer Tripel zum Bau perfekter, spannungsfreier Winkel
- Erweiterte Startblockausrichtung durch anfängliches Fahren entgegengesetzt zur Bewegungsrichtung, um durch Spiel in Motoren ausgelöste Ungenauigkeiten in der Positionierung auszugleichen
- Individueller Startzeitpunkt zum Suchen nach Linien, bspw. zum Überspringen früherer Linien
- Möglichkeit für Drehungen über 180°

Verbesserungen seit dem Regionalwettbewerb

- Verbesserungen an der Zuverlässigkeit der Runden
- Beschleunigung der bestehenden Runden
- Optimierung der Geschwindigkeitsberechnung
- Großes Update am öffentlichen Repository