

## 3 Umgang mit dem Dobot Magician

### Inhalt

<b>3</b>	<b>Umgang mit dem Dobot Magician</b>	<b>55</b>
3.1	Grundlagen zum Dobot Magician	56
3.2	Teaching & Playback	59
3.3	Schreiben & Zeichnen	62
3.4	Grundlagen in Blockly	64
	3.4.1 Pick & Place mit Variablen	64
	3.4.2 Ablagelogiken	70
3.5	Lichtschanke, Farbsensor und Sortierung	73
	3.5.1 Einbindung des Förderbandes	73
	3.5.2 Einbindung der Lichtschanke	76
	3.5.3 Farbsensor	80
3.6	Linearachse	82
	3.6.1 Verbindung mit der Linearachse	82
	3.6.2 Steuerung der Linearachse	82
	3.6.3 Interaktion mit dem Dobot und der Linearachse	83
3.7	Nutzung der IO-Ports zur Vernetzung	86
	3.7.1 Vernetzung mittels Taster	86
	3.7.2 Vernetzung mittels EIO-Schnittstelle	86
	3.7.3 Einführung in Python	90
3.8	Vergleich der Programmieroptionen, Möglichkeiten in DobotLab	99
	3.8.1 Allgemeine Vorstellung DobotLab	99
	3.8.2 Python Lab – Skriptbasierte Programmierumgebung	101
	3.8.3 Writing and Drawing	101
	3.8.4 Fehlerliste der Hardware Dobot Magician und der Softwareanwendungen DobotStudio, DobotBlock und DobotLab	102
	3.8.5 Vorteile bei der Nutzung von DobotLab	103

### 3.1 Grundlagen zum Dobot Magician

Rene Egbers, Fabian Icken, Marcus Auf der Landwehr, Hochschule Osnabrück

Der *Magician* des chinesischen Herstellers *Dobot* ist ein 4-Achs-Desktop-Roboter, der für die Aus- und Weiterbildung konzipiert wurde. Im Rahmen der Robonatives Förderung haben sich der überwiegende Teil allgemeinbildender Schulen und vereinzelt auch berufsbildende Schulen für die Anschaffung dieser Geräte entschieden. Die Popularität des *Magicians* lässt sich darauf zurückführen, da es sich bei diesem Gerät um eine vielseitige und im Vergleich zu industriellen Systemen deutlich kostengünstigere Variante handelt, mit der sich viele Inhalte des Unterrichts visualisieren und erlebbar machen lassen. Zur Bezeichnung des *Magicians* wird häufig auch simultan die Bezeichnung *Dobot* verwendet. Neben dem eigentlichen Roboter, bietet der Hersteller umfangreiches Zubehör an. Zusätzlich erworben werden können ein Förderband, eine Linearachse, ein Farbsensor, ein Reflexlichttaster (vom Hersteller als photoelektrischer Sensor bezeichnet), ein Kamerasystem (vom Hersteller als Vision Kit bezeichnet) und ein Arduino-KI-Kit (vom Hersteller als Basic-AI-Kit bezeichnet). Im Lieferumfang eines einzelnen Roboters sind neben einem Parallelbacken- und einem Sauggreifer, einem Kompressor, einem Stifthalter und einigen Schaumstoffwürfeln auch Zubehörteile für das 3D-Drucken mit dem *Magician* enthalten. Der Roboter unterstützt ein Handhabungsgewicht von maximal 500 g und erreicht eine Wiederholungsgenauigkeit von 0,2 mm. Die folgende Abbildung zeigt einen *Magician* mit montiertem Parallelbackengreifer.



Abbildung 1: Dobot Magician mit Parallelbackengreifer

Der *Magician* ist mit einer Parallelkinematik ausgestattet, was es nur ermöglicht, Punkte „von oben“ anzufahren. Der Endeffektor kann nur um seine z-Achse gedreht werden, jedoch nicht um seine x- oder y-Achse. Die folgenden Darstellungen zeigen neben den Gelenken des Roboters auch die Lage des Basis-Koordinatensystems. Alle Koordinaten im Arbeitsraum des Roboters werden auf diesen Punkt bezogen angegeben. Dieser Nullpunkt liegt zwischen den Antrieben der Parallelkinematik und mittig in der Roboterbasis, kann aber mit dem Endeffektor nicht selbst angefahren werden. Rund um die Basis gibt es einen Sperrbereich der aufgrund der Kinematik nicht erreichbar ist.

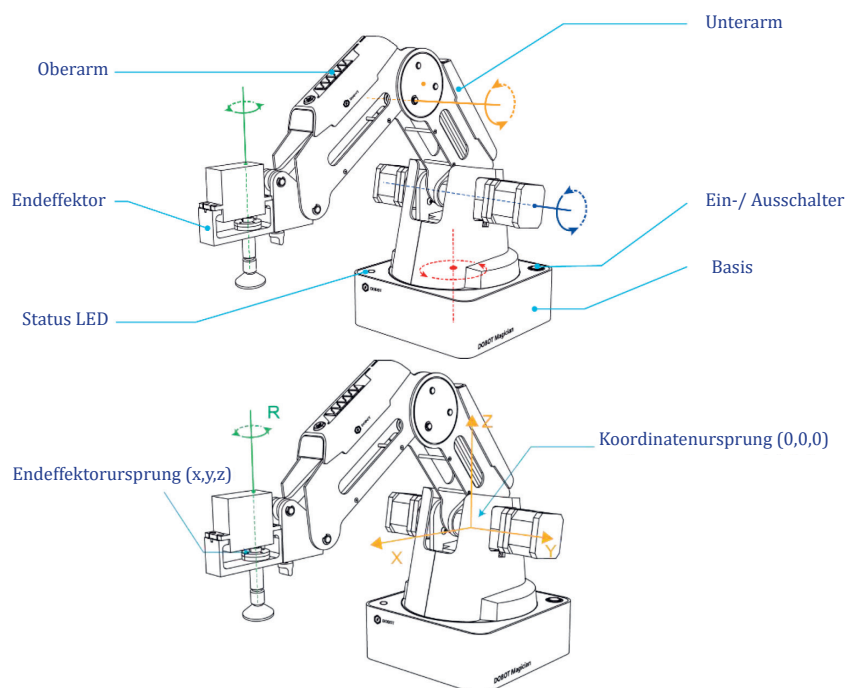


Abbildung 2: Koordinatensysteme und Aufbau des Dobot Magician

Mittlerweile werden drei Computerprogramme zur Programmierung des Magicians angeboten, die von der Webseite des Herstellers kostenfrei heruntergeladen werden können. Das originäre Programm heißt DobotStudio, danach wurde das Programm DobotBlock und vor kurzer Zeit wurde die dritte Software namens DobotLab vorgestellt. Zu Beginn des Projektes war nur DobotStudio verfügbar, weshalb die Lehrkräfte in den Projektschulen ebenfalls im Umgang mit dieser Software geschult wurden und diese Software auch bis heute im Unterricht einsetzen. DobotStudio enthält bis zuletzt kleinere Softwarebugs, die in der Anwendung aber nicht wesentlich stören. Vorteilhaft an DobotStudio ist, dass hier abgesehen vom 3D-Druck alle Funktionen, die der Dobot unterstützt, in einem Programm zusammengeführt sind. Die folgenden Ausführungen beziehen sich immer auf die Software DobotStudio. Ein Vergleich von DobotStudio und dem etwas neuerem Programm DobotBlock kann im Abschnitt „Vergleich der Programmieroptionen“ dieses Werks gefunden werden.

Die folgende Abbildung zeigt den Startbildschirm von DobotStudio.

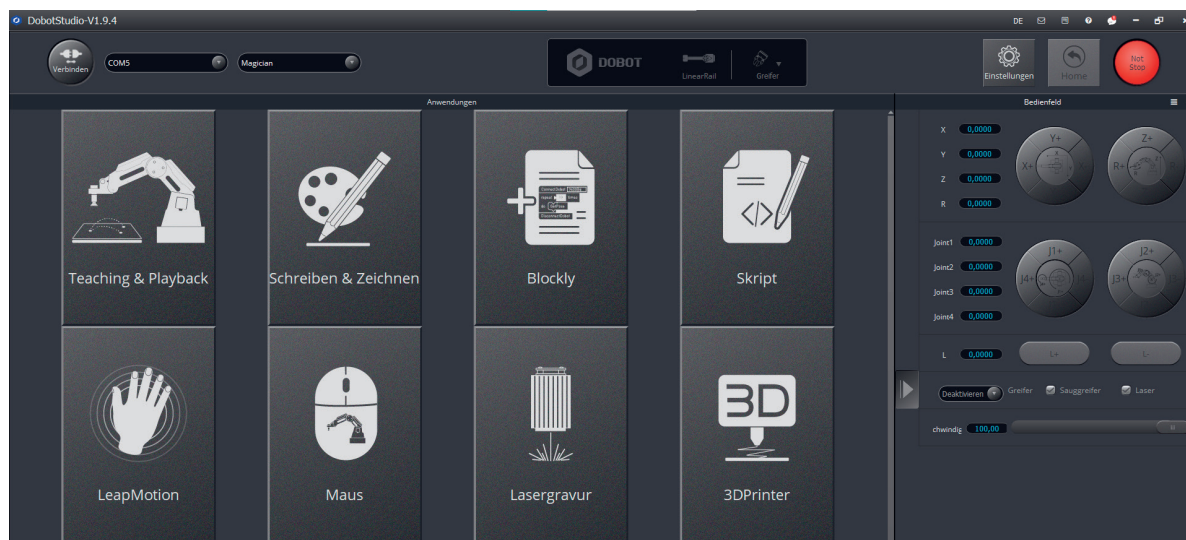


Abbildung 3: Dobot Studio Startbildschirm

In der linken oberen Ecke befinden sich die Schaltflächen, um eine Verbindung zwischen DobotStudio und dem Magician herzustellen. Zunächst muss der COM-Port ausgewählt werden, über den der Roboter mit dem PC verbunden ist. Stehen hier mehrere Ports zur Auswahl, kann herausgefunden werden, welcher zum Magician gehört, indem er per USB-Kabel getrennt und wieder verbunden wird und man beobachtet, welcher COM-Port verschwindet und wieder erscheint. Den entsprechenden COM-Port dann auswählen und überprüfen, ob im Drop-down-Menü rechts daneben der Typ „Magician“ eingestellt ist. Anschließend kann über den runden Button links eine Verbindung mit dem Magician hergestellt werden. Konnte eine Verbindung hergestellt werden, wechselt der Button-Text zu „Trennen“ und auf der rechten Seite werden die runden Steuerflächen aktiv. Konnte keine Verbindung hergestellt werden, erscheint eine entsprechende Fehlermeldung.



Abbildung 4: Schaltfläche zur Herstellung einer Verbindung

Ebenfalls möglich ist, dass eine Verbindung aufgebaut werden konnte, jedoch in Anzeige oben in der Mitte eine Fehlermeldung angezeigt wird. Diese kann mithilfe eines Doppelklicks geöffnet werden und nach Beseitigung mittels der Taste „Clear Alarm“ quittiert und gelöscht werden. Wenn der Magician mittels der Freedrive-Taste anschließend aus dem Sperrbereich direkt am Roboter heraus bewegt wurde, sollte hier keine Fehlermeldung mehr auftauchen und die Signalleuchte am Roboter in grüner Farbe dauerhaft leuchten. Ebenfalls in diesem Feld befinden sich die Schaltflächen, um den Endeffektor und, falls angeschlossen, die Linearachse auszuwählen. Nach der Aktivierung ist die Schaltfläche blau hinterlegt.

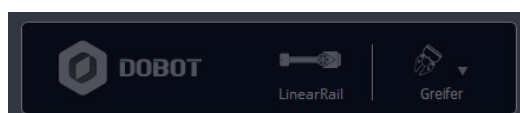


Abbildung 5: Schaltfläche zur Aktivieren von Endeffektoren und der Linearachse

Neben dem Reiter zur Herstellung einer Verbindung und zur Auswahl eines Endeffektors, befindet sich in der oberen Leiste ein Reiter, auf dem die Einstellung verlinkt sind und auf dem ein Not-Stopp ausgelöst werden kann.

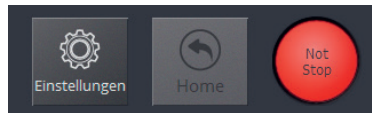


Abbildung 6: Buttons für Einstellungen, Homing und Not-Halt

Zwischen den Einstellungen und dem Not-Stopp befindet sich eine Schaltfläche, die mit „Home“ bezeichnet ist. Durch Betätigung des Home-Buttons wird eine automatische Referenzfahrt gestartet.

Über das Bedienfeld kann der Magician gesteuert werden. Möglich ist eine Bewegung in kartesischen Koordinaten in x-, y- und z-Richtung und die Rotation R des Endeffektors. Alternativ können direkt die Gelenkwinkel Joint1, Joint2, Joint3 und Joint4 des Roboters gesteuert werden. Wenn eine Linearachse angeschlossen ist, kann diese mithilfe der Buttons „L+“ und „L-“ manuell verfahren werden. Mit den Schaltflächen für „Greifer“, „Sauggreifer“ und „Laser“ können der jeweilige Endeffektor betätigt werden. Ebenfalls lässt sich in diesem Reiter die prozentuale Geschwindigkeit des Roboters einstellen.



Abbildung 7: Buttons für Einstellungen, Homing und Not-Halt

Auf dem Startbildschirm von DobotStudio sind alle Applikationen verknüpft, die mit dem Magician genutzt werden können. Auf die wichtigsten Anwendungen wird hier kurz eingegangen:

1. **Teaching & Playback:** Hierbei handelt es sich um eine Programmierart, bei der Punkte bei einer Handführung eingelernt werden (auch einteachen genannt), die anschließend vom Magician abgefahren werden. An den Punkten können Endeffektorbefehle, z. B. öffnen und schießen des Greifers, hinzugefügt werden. Diese Art der Programmierung wird im nächsten Abschnitt genauer erklärt.
2. **Schreiben & Zeichnen:** Mit dieser Applikation kann der Magician Text schreiben und Bilder zeichnen. Im Teil „Einführung in das Schreiben & Zeichnen“ wird diese Funktion genauer beschrieben.
3. **Blockly:** Die Programmierart Blockly ermöglicht eine blockbasierte Programmierung des Magicians. Im Kapitel „Einführung in Blockly“ wird diese Programmierart genauer beschrieben.
4. **Skript:** Die Programmierart Skript ermöglicht die Programmierung des Magicians mit Python. Im Kapitel „Einführung in Python“ wird diese Programmierart genauer beschrieben.
5. **Lasergravur:** Die Funktion Lasergravur soll das Gravieren von Gegenständen mit dem Dobot ermöglichen. Der Laser und dessen Einfuhr ist jedoch in Deutschland aus Sicherheitsgründen verboten.
6. **3D Printer:** Die 3D-Druck Funktion ermöglicht es, mit dem Dobot 3D zu drucken. Hierfür muss eine andere Firmware auf den Magician geladen werden. Auch wird eine zusätzliche Slicer-Software benötigt. Unterstützt werden der Cura- und der Repetier-Slicer. Eine detaillierte Anleitung ist in den Handbüchern von Dobot enthalten.

## 3.2 Teaching & Playback

Rene Egbers, Fabian Icken, Marcus Auf der Landwehr, Hochschule Osnabrück

Das Teaching & Playback ermöglicht die Programmierung des Dobot durch das Setzen von Punkten, die der Roboter beim Start des Programms nacheinander abfährt. An den jeweiligen Punkten können dem Dobot noch gewisse Tätigkeiten, wie das Aktivieren und Deaktivieren des Endeffektors zugewiesen werden. Beim Öffnen der Teaching & Playback-Programmierart, erscheint zunächst die folgende Oberfläche:

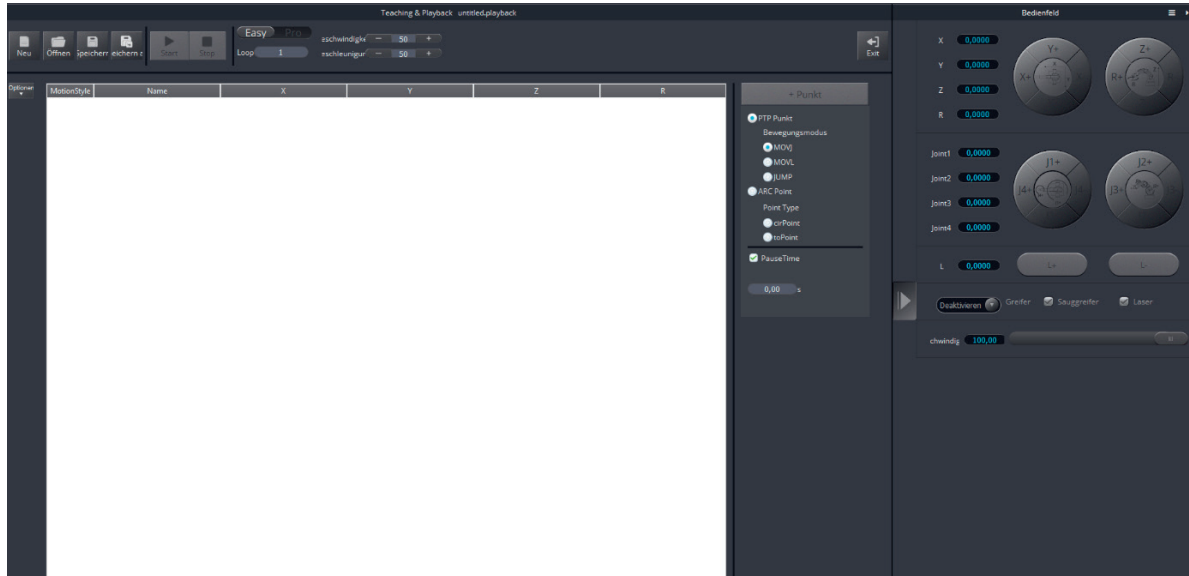


Abbildung 1: Programmieroberfläche der Anwendung Teaching & Playback

Zu sehen ist, dass die Bedienoberfläche an der rechten Seite so bleibt wie im Startbildschirm der Software. Lediglich das Fenster für die Anwendungen ändert sich. Hier erscheint die Teaching & Playback Bedienoberfläche. Diese besteht aus der oberen Leiste mit verschiedensten Funktionen, der rechten Leiste zur Festlegung der Bewegungsart und dem Programmierfeld.

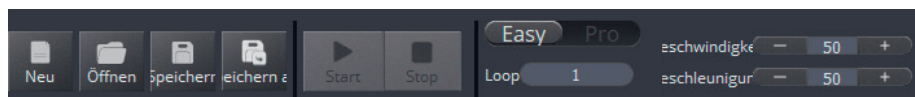


Abbildung 2: Schaltfläche mit verschiedenen Funktionen in der Anwendung Teaching & Playback

In der oberen Leiste tauchen, wie in Abbildung 2 zu sehen, mehrere Funktionen nebeneinander auf. Links sind Optionen verzeichnet, zum Starten neuer Programme, zum Öffnen von Programmen und zum Speichern von Programmen. Daneben sind die Schaltflächen zum Starten und Stoppen des Programmablaufs. Gefolgt von einer Schaltfläche zum Wechsel des Programmiermodus. Hier kann zwischen Easy und Pro unterschieden werden. Unter dieser Schaltfläche befindet sich die Schaltfläche „Loop“, hier kann die Anzahl der Wiederholungen des Programmablaufs festgelegt werden. An der rechten Seite dieser Leiste befindet sich eine Schaltfläche, mit der die Geschwindigkeit und Beschleunigung des Magicians in % reguliert werden können.

Die Leiste mit den Bewegungsarten sieht wie folgt aus.

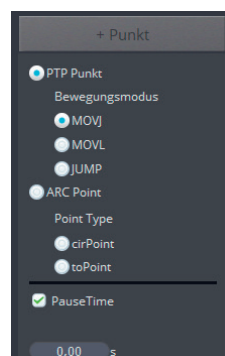


Abbildung 3: Bedienfeld zur Auswahl der Bewegungsart

Hier kann zwischen *PTP Punkt* und *ARC Point* unterschieden werden. Ebenfalls kann eine Pausenzeit (*PauseTime*) an den jeweiligen Punkten festgelegt werden. Unter *PTP Punkt* sind mehrere Bewegungsmodi verzeichnet. Diese werden im Folgenden genauer erörtert.

### Bewegungsmodus MOVJ

Der Bewegungsmodus MOVJ bildet eine gängige PTP-Bewegung ab. Bei einer PTP-Bewegung werden die Anfangs- und Endpunkte einprogrammiert, zwischen denen sich der Roboter bewegen soll. Die Steuerung berechnet einmalig die benötigten Gelenkwinkel, die der Roboter zum Erreichen seines Endpunktes anfahren muss und bewegt die Antriebe genau in diese Position. Im industriellen Bereich sind diese Bewegungen schneller als die anderen Bewegungsarten. Es kann jedoch zu einem unvorhersehbaren Bewegungsablauf kommen. Ein Kennzeichen für das Verfahren im MOVJ-Bewegungsmodus bzw. in einer PTP-Bewegung ist eine eher kreisförmige Bewegung des Roboters.

### Bewegungsmodus MOVL

Der Bewegungsmodus MOVL bildet hingegen eine gängige Linearbewegung ab. Bei einer Linearbewegung werden wie bei der PTP-Bewegung Anfangs- und Endpunkt einprogrammiert. Die Steuerung versucht nun durch eine Interpolation eine möglichst gerade Linie zwischen den beiden Punkten abzufahren. Hierfür werden Punkte auf der geraden Linie zwischen den Bahnen berechnet und in einem bestimmten Interpolationstakt (dieser liegt im Millisekundenbereich) abgefahren. Hierbei gilt, je geringer der Interpolationstakt, umso genauer fährt der Roboter die Bahn ab. Es entsteht eine annähernd gerade Bahn zwischen dem Anfangs- und dem Endpunkt, die der Roboter abfährt. Abbildung 4 zeigt die schematische Darstellung einer Linearbewegung.

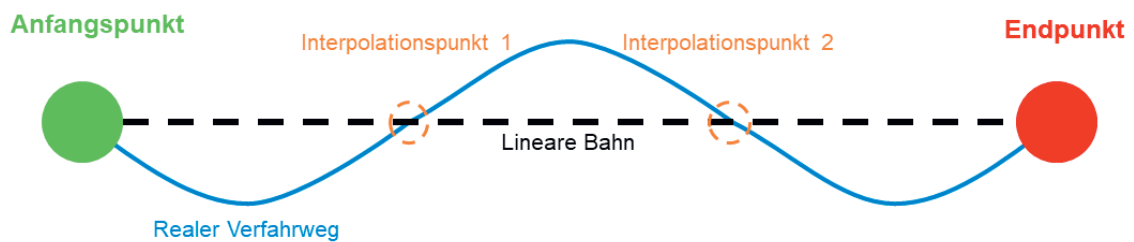


Abbildung 4: Schematische Darstellung einer Linearbewegung

### Bewegungsmodus JUMP

Der Bewegungsmodus JUMP ist eine Kombination aus einer PTP-Bewegung und einer Linearbewegung. Es werden, wie bei den beiden anderen Bewegungsmodi, ein Anfangs- und ein Endpunkt einprogrammiert. Der Dobot fährt vom Anfangspunkt eine gewisse vorher eingestellte Distanz  $\Delta h$  (voreingestellt sind ca. 2cm) mit einer Linearbewegung (MOVL) nach oben, fährt mit einer PTP-Bewegung (MOVJ) auf eine Distanz  $\Delta h$  über den Endpunkt und von hier fährt der Roboter wieder mit einer Linearbewegung (MOVL) nach unten. Abbildung 5 zeigt diesen Bewegungsmodus schematisch.

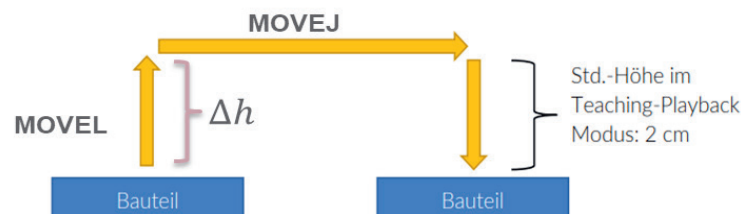


Abbildung 5: Schematische Darstellung einer JUMP-Bewegung

## Programmieren mit Teaching & Playback

Für das Programmieren mit Teaching & Playback empfiehlt es sich zuerst eine Referenzfahrt mit „Home“ durchzuführen und den Endpunkt der Referenzfahrt als ersten Programmpunkt festzulegen. Dies gelingt durch das Klicken auf die Schaltfläche „+ Punkt“. Das Ganze sieht dann wie in Abbildung 6 aus. Die Bewegungsart kann für jeden Wegpunkt einzeln festgelegt werden, indem diese unter „MotionStyle“ geändert wird. Wird die Bewegungsart auf der Leiste für die Bewegungsarten eingestellt, so gilt diese übergeordnet für alle nachfolgend einprogrammierten Punkte. Es kann ebenfalls jedem Punkt, wie in Abbildung 6 zu sehen im rechten Feld der Programmieroberfläche ein Befehl zugeordnet werden, mit dem der Endeffektor aktiviert oder deaktiviert wird.

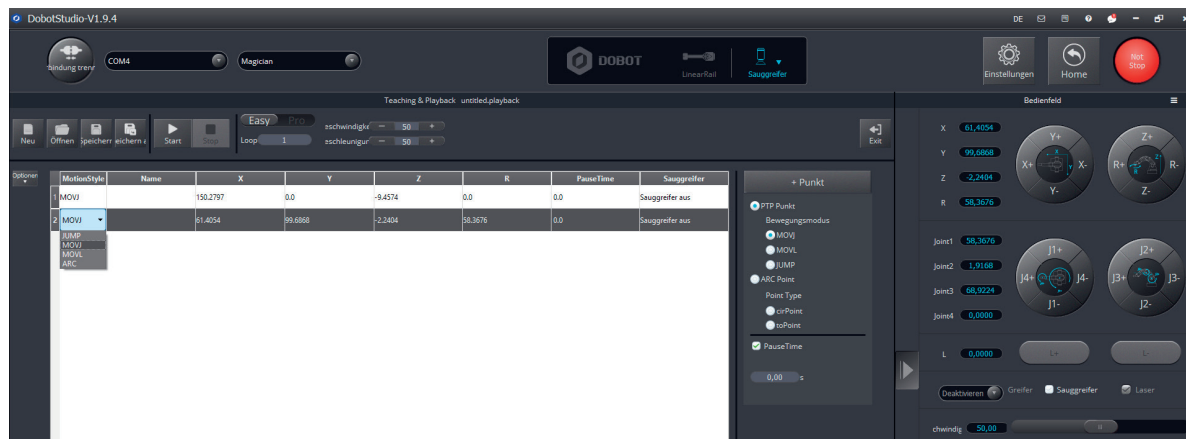


Abbildung 6: Schematische Darstellung einer Linearbewegung

Ist der erste Punkt festgelegt, so kann der Roboter entweder über das Bedienfeld oder über eine Taste, die mit einem Schloss gekennzeichnet ist und sich direkt vorne am Roboterarm befindet, zu einem nächsten Punkt bewegt werden.

**Wichtig:** Ist im Programm vorgesehen, dass der Endeffektor mit seiner vierten Achse (über R steuerbar) rotiert, so empfiehlt sich die Bewegung des Roboters über das Bedienfeld, da nur hierüber die jeweilige R-Position im übergeordneten Skript hinterlegt wird und so Wiederholgenauigkeit im Programm erreicht werden kann. Wird der Roboter über die Schloss-Taste programmiert, so kommt es hierbei zu Fehlern und die Rotation des Endeffektors ist bei jedem Programmablauf anders ausgerichtet.

Das gesamte Programm wird nun aus einer Aneinanderreihung von Punkten erstellt, die vom Roboter nacheinander abgefahren werden. Jedem Punkt kann ein Befehl für den Endeffektor oder ein Schaltbefehl für einen EIO-Ausgang hinzugefügt werden.

Teaching & Playback bietet eine einfache Art der Programmierung, besonders für die Anfangssituationen, in denen Schülerinnen und Schüler den Roboter kennenlernen. Bei Messebesuchen oder sonstigen Veranstaltungen zeigte sich, dass selbst Grundschüler\*innen innerhalb weniger Minuten erste eigene Programme mit dieser Art der Programmierung schreiben können.

### Home-Position ändern

Die Teaching & Playback Programmieroberfläche bietet die einzige Möglichkeit, die Home-Position zu ändern, ohne dass die Änderung nach dem Abschalten des Dobot wieder rückgängig gemacht wird. Dies gelingt durch das Einprogrammieren eines Punktes, welcher die neue Home-Position darstellen soll. Ist der Punkt einprogrammiert, so werden durch einen Rechtsklick auf den Punkt die folgenden in Abbildung 7 dargestellten Optionen angezeigt. Unten ist die Option „Home setzen“ aufgeführt. Mit einem Klick auf diese Option erscheint ein Textfeld, in dem „Home Position erfolgreich gesetzt!“ steht. Jetzt fährt der Roboter am Ende der Referenzfahrt (Home) immer an die neu einprogrammierte Position.

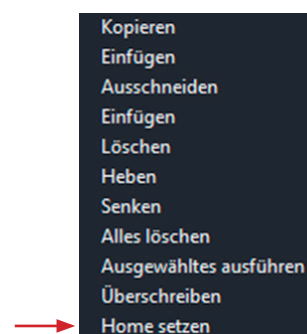


Abbildung 7: „Home setzen“

### 3.3 Schreiben & Zeichnen

Rene Egbers, Fabian Icken, Marcus Auf der Landwehr, Hochschule Osnabrück

Das Anwendungsfeld des Schreibens & Zeichnen in der Startoberfläche des DobotStudios ermöglicht es, mithilfe des Dobot zu schreiben und Bilder zeichnen zu lassen. Im Folgenden werden die notwendigen Schritte für die „Programmierung“ des Dobot genauer erläutert.

Für diese Anwendung wird der im Lieferumfang des Magicians enthaltene Stifthalter benötigt, der vorne in die Aufnahme für die Endeffektor montiert wird. In den Stifthalter selbst sollte anstelle des mitgelieferten Fineliners ein Bleistift gespannt werden, da der Fineliner bei längerem Kontakt mit dem Papier zu größeren Flecken führt. Bei der Montage ist zu beachten, dass der Stift nicht zu weit nach oben ragt, da er sonst bei einer Konturfahrt dicht an der Basis selbst die Freedrive-Taste des Roboters betätigt. Gleichzeitig sollte der Stift nicht zu hoch eingespannt werden, da sich dieser sonst aufgrund der Nachgiebigkeit des Stiftes Vibrationen ergeben, die zu unsauberen Zeichenergebnissen führen können.

Nach der erfolgreichen Montage von Stift und Stifthalter kann die Applikation Schreiben & Zeichnen im Programm DobotStudio gestartet werden. Zu Beginn muss der Stift als Endeffektor ausgewählt werden. Dies erfolgt über das oben mittige Drop-Down-Menü (siehe Pfeil in der folgenden Abbildung).

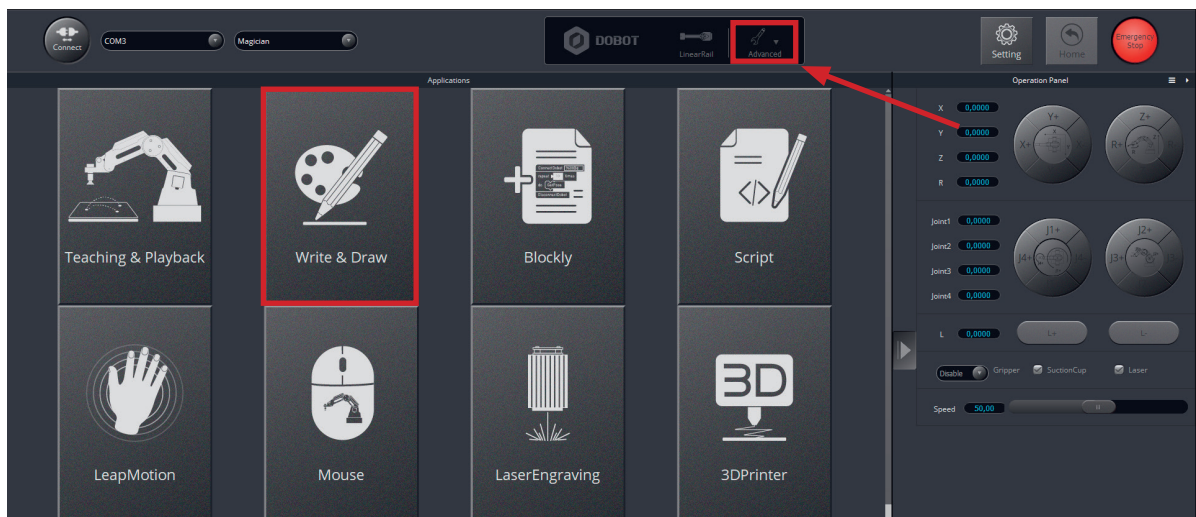


Abbildung 1: Starten der Applikation Schreiben & Zeichnen

Es öffnet sich die in der folgenden Abbildung zu sehende Programmoberfläche. Die aus zwei Halbkreisabschnitten bestehende Darstellung visualisiert den Arbeitsbereich des Magicians, also Punkte, die mit dem Stift erreicht werden können. Wenn der Magician mit der Software verbunden ist, wird die aktuelle Position des Endeffektors in dieser Darstellung mit einem größeren grauen Punkt angezeigt.

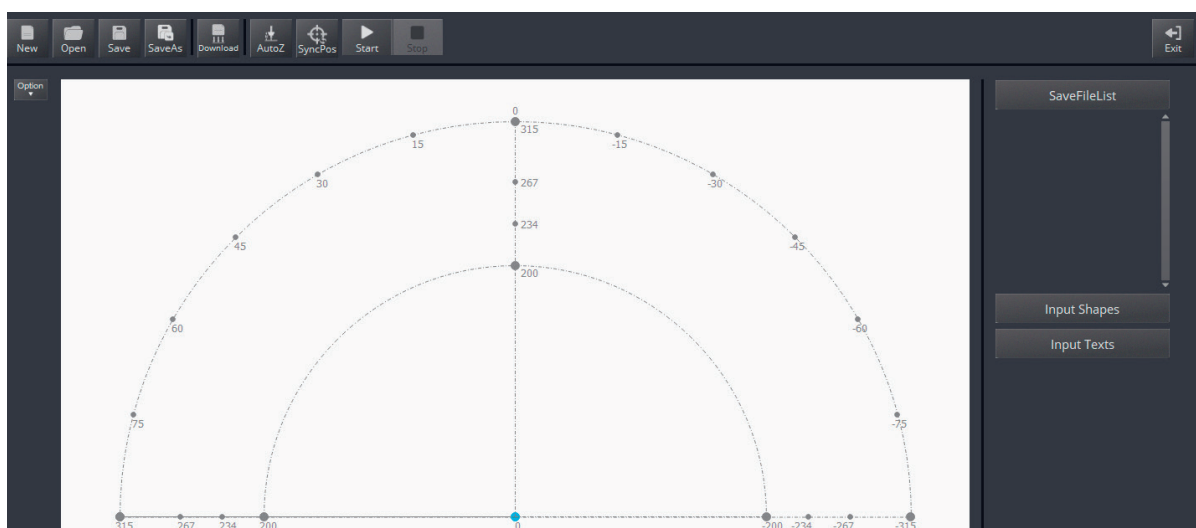


Abbildung 2: Applikation Schreiben & Zeichnen



Bevor mit dem Schreiben & Zeichnen begonnen werden kann, sollte ein Blatt Papier vor dem Magician auf dem Tisch befestigt werden. Für die Praxis kann die Papiergröße DIN-A3 empfohlen werden.

## Bilder importieren

Damit der Magician Bilder nachzeichnen kann, müssen diese in die Zeichenfläche importiert werden. Dies erfolgt über die Schaltfläche *Öffnen*. Zu beachten ist, dass nur das Zeichnen von Vektorgrafiken (z. B. SVG-Dateien) unterstützt werden.

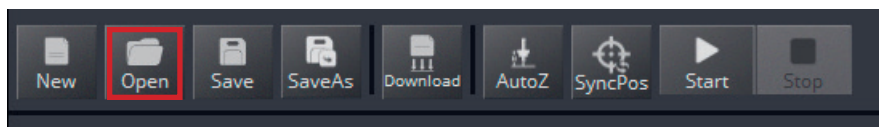


Abbildung 3: Schaltfläche in der Anwendung Schreiben & Zeichnen

Wurde nun ein Bild ausgewählt und geöffnet, erscheint dieses in der Zeichenfläche und kann dort skaliert und verschoben werden. Zu beachten ist, dass das Bild in den Arbeitsbereich, zwischen den beiden Radien, platziert werden muss, ansonsten färbt sich das Bild rot und es kann nicht gezeichnet werden. Alternativ zu den Bildern können auch die Symbole und Bilder aus der Dobot Software verwendet werden, indem auf „Input Shapes“ am rechten Rand geklickt wird und dort die passenden Symbole und Bilder ausgewählt werden.

Sollen andere Bildformate verwendet werden, ist dies nur möglich, indem diese zuvor umgewandelt werden. Dafür bietet DobotStudio die Funktion „Convert Bitmap“ am rechten Rand des Bildschirms. Anschließend kann das Bild mithilfe des Buttons „Plot to Main Scene“ in der Zeichenfläche platziert werden.

Neben Bildern und Symbolen kann der Magician auch Text schreiben. Am rechten Bildschirmrand gibt es dafür die Eingabemaske „Input Text“. Durch klicken auf „OK“ wird der Text in die Zeichenfläche eingefügt. Auch hier ist wieder eine Skalierung und Verschiebung der Textelemente möglich. Neben der Schriftart kann der Schriftschnitt über die entsprechenden Buttons verändert werden.

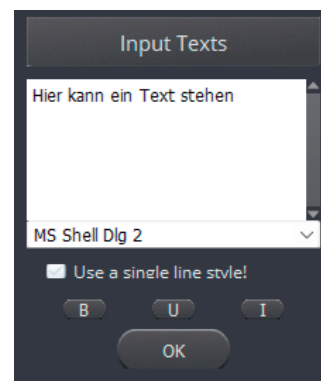


Abbildung 5: Input Text - Schreiben & Zeichnen

Ist der Arbeitsbereich mit den gewünschten Bildern, Symbolen und Texten angereichert worden, kann der Schreibvorgang gestartet werden. Zunächst muss dafür einmalig die Referenzhöhe eingelernt werden. Dafür muss der Roboterarm über die Steuerflächen oder händisch mithilfe der Freedrive-Taste so weit nach unten auf das Papier gefahren wird, sodass die Stiftspitze das Blatt mit einem leichten Druck berührt.

Damit die Referenzhöhe gesetzt wird, muss anschließend die Taste „AutoZ“ betätigt werden. Ist dies geschehen, kann der Schreibvorgang über die Start-Taste gestartet werden. Während des Schreibvorganges kann dieser auch gestoppt oder abgebrochen werden. Zeigt sich, dass die Stiftspitze nicht in allen Bereichen der Zeichenfläche die Papierebene berührt, kann dies durch eine erneute Kalibrierung mithilfe des Auto-Levelling-Tools behoben werden. Eine schrittweise Anleitung hierzu kann in den Einstellungen von DobotStudio gefunden werden.

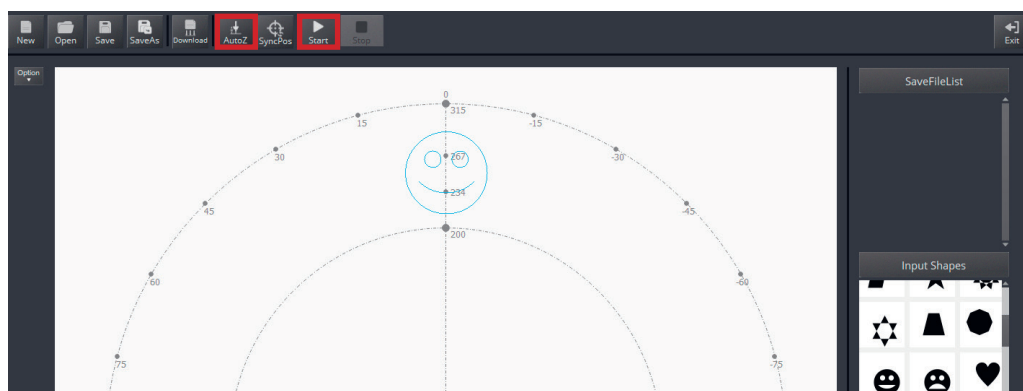


Abbildung 6: AutoZ und Start in der Anwendung Schreiben & Zeichnen

## 3.4 Grundlagen in Blockly

Dani Hamade, Jan Landherr, Carl von Ossietzky Universität Oldenburg

### 3.4.1 Pick & Place mit Variablen

In diesem Abschnitt soll eine kurze Einführung in die blockbasierte Programmierung des Dobots erfolgen. Hierzu kann eine erste Übung durchgeführt werden, in welcher ein Würfel, welcher sich in der Ausgangsposition A befindet, zur Endposition B befördert werden soll (siehe Abbildung 1).

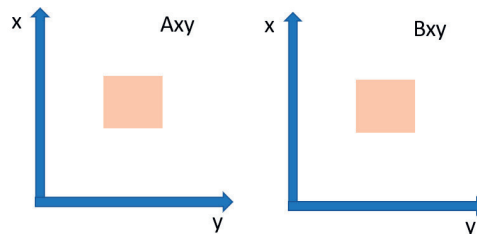


Abbildung 1: Erste Übung: Würfel Axy soll nach Bxy befördert werden

Ein Ansatz, welcher hier verfolgt werden kann, ist die Definition der Positionen durch Variablen. Hier werden die Koordinaten für die Position A beispielsweise mit Aufnahme X, Y und Z definiert. Punkt A wäre somit definiert und es muss lediglich noch ein Ablagepunkt B definiert werden. Auch hierzu werden Variablen definiert, die mit Ablage X, Y und Z beschriftet werden. Für eine bessere Übersichtlichkeit können alle Variablen innerhalb eines Funktionsblocks, welcher mit „Variablen“ beschriftet wird, abgelegt werden. Hierzu kann man wie folgt vorgehen. Man navigiert in den Programmblöcken bei Blockly in DobotStudio auf den Reiter „Funktionen“ und zieht einen einfachen Funktionsblock in das Programm (siehe Abbildung 2).

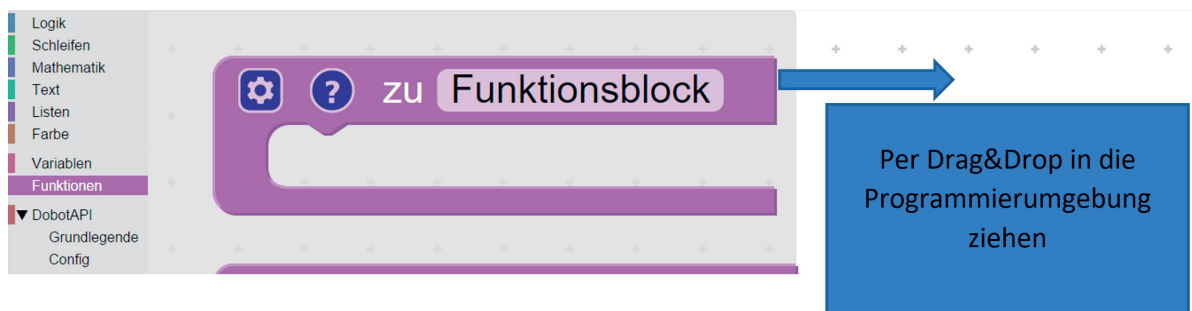


Abbildung 2: Wählen eines Funktionsblocks in DobotStudio Blockly

Anschließend benennt man diesen durch doppeltes Klicken auf das Schriftfeld (Doppelklick auf den Begriff „Funktionsblock“) um in „Variablen“ (siehe Abbildung 3).

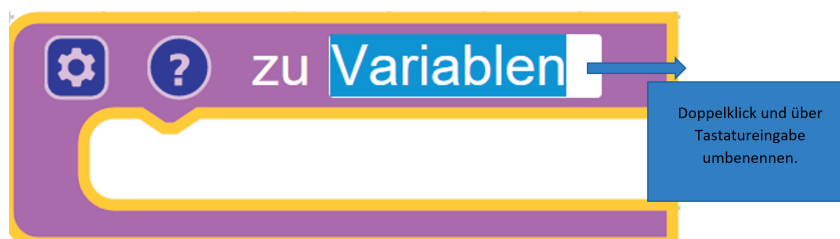


Abbildung 3: Umbenennen von Funktionsblöcken

Nachdem dieser Schritt abgeschlossen ist, kann mit der Definition der Variablen begonnen werden. Hierzu wählt man in den Programmblöcken den Reiter „Variablen“ an. Anschließend zieht man einen Block „Schreibe Element“ wieder per Drag&Drop in die Programmieroberfläche (siehe Abbildung 4). Der Befehl „Schreibe Element“ erlaubt es nun, neue Variablen zu definieren.

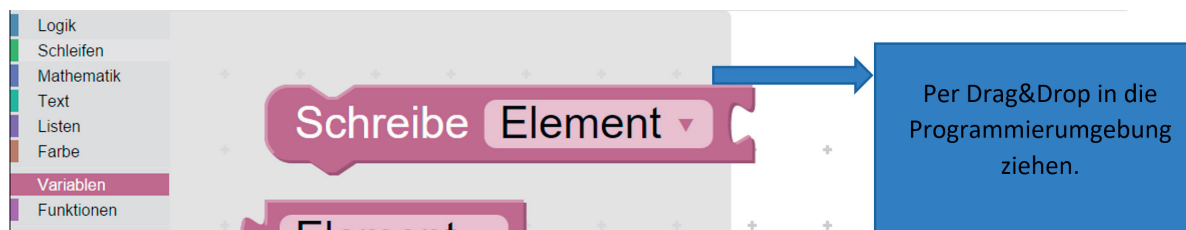


Abbildung 4: Wählen eines Variablenblocks

Man zieht den Block „Schreibe Element“ hierzu nun in den Funktionsblock „Variablen“, um mit der Definition von Variablen zu beginnen (siehe Abbildung 5). Damit die Variable nun individuell definiert werden kann, öffnet man das Dropdown Menü bei dem Block „Schreibe Variable“ und klickt auf „Neue Variable...“ (siehe Abbildung 6).

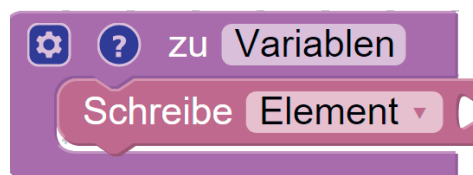


Abbildung 5: Definition neuer Variablen

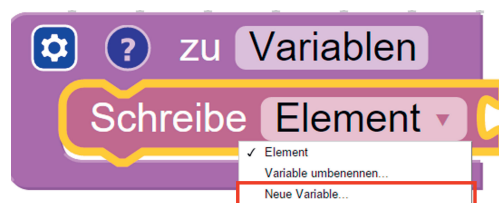


Abbildung 6: Anlegen neuer Variablen

Es öffnet sich nun ein Schriftfeld, in dem man der Variable eine Bezeichnung geben kann. Für die Aufgabenstellung werden insgesamt sechs Variablen benötigt, nämlich für jede Koordinate eine (Aufnahme X, Y und Z und Ablage X, Y und Z). Die erste Variable wird demnach „AufnahmeX“ benannt.

Durch Drücken auf „OK“ wird die Variable in der Ansicht umbenannt (siehe Abbildung 7).

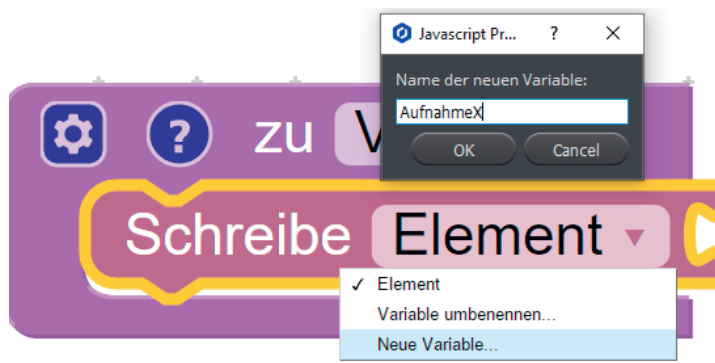


Abbildung 7: Benennen von Variablen

Es öffnet sich nun ein Schriftfeld, in dem man der Variable eine Bezeichnung geben kann. Für die Aufgabenstellung werden insgesamt sechs Variablen benötigt, für jede Koordinate eine (Aufnahme X, Y und Z und Ablage X, Y und Z). Die erste Variable wird demnach „AufnahmeX“ benannt.

Durch Drücken auf „OK“ wird die Variable in der Ansicht umbenannt (siehe Abbildung 7).

Diesen Schritt wiederholt man nun für alle notwendigen Variablen. Hierzu kann man entweder wieder über den Programmblock „Variablen“ neue Bausteine in die Funktion ziehen (siehe Abbildung 4) oder über Copy&Paste die bereits definierte Variable kopieren und einfügen (hierzu die Variable anwählen, sodass diese gelb aufleuchtet, dann Strg & C und Strg & V.

HINWEIS: Die eingefügte Variable muss über das DropDown Menü wieder überschrieben werden!). So sieht es aus, wenn alle sechs Variablen definiert sind:

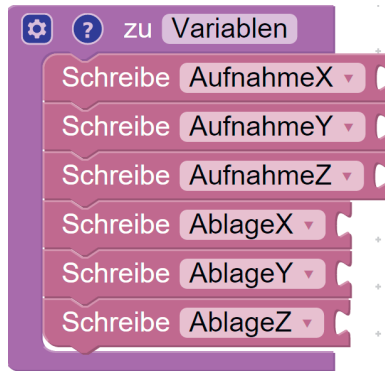


Abbildung 8: Ansicht Funktionsblock Variablen

Den einzelnen Variablen müssen nun noch entsprechende Koordinaten (Werte) zugeordnet werden. Hierzu wählt man bei den Programmblöcken den Reiter „Mathematik“ aus und zieht eine einfache Zahl (siehe Abbildung 10) per Drag&Drop an die einzelnen Variablen im Funktionsblock (Hinweis: Die Zahlenblöcke müssen an die einzelnen Variablen „andocken“). Diesen Vorgang wiederholt man für alle sechs Variablen. Ist dieser Schritt erfolgt, kann man nun die passenden Werte der Koordinaten übertragen. Hierzu bewegt man den Dobot-Arm zu der entsprechenden Position (Aufnahme X, Y und Z bilden in der Aufgabenstellung den Punkt A). Man drückt und hält hierzu den Entriegelungsknopf am Arm des Dobots und führt ihn an die Würfeloberfläche. Anschließend kann man die Koordinaten am Steuerkreuz auf der rechten Seite des Bildschirms ablesen und übertragen (siehe Abbildung 9). Den Vorgang wiederholt man für die Ablagekoordinaten, sodass allen Variablen Werte zugeordnet werden können (siehe Abbildung 11).



Abbildung 9: Ablesen der Koordinaten am Steuerkreuz

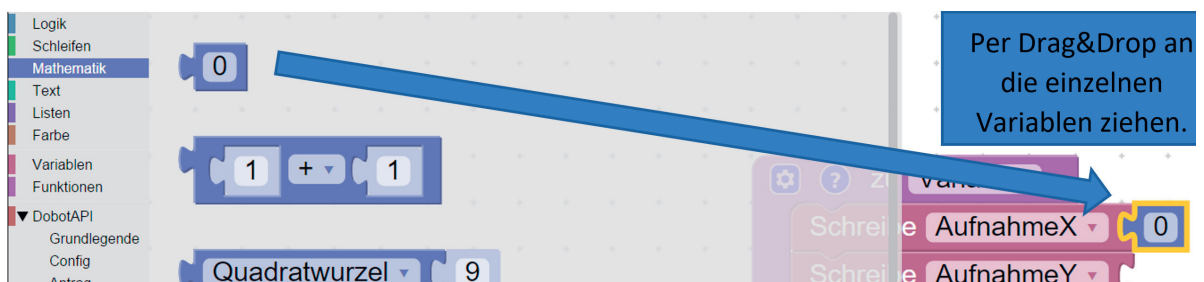


Abbildung 10: Ablesen der Koordinaten am Steuerkreuz

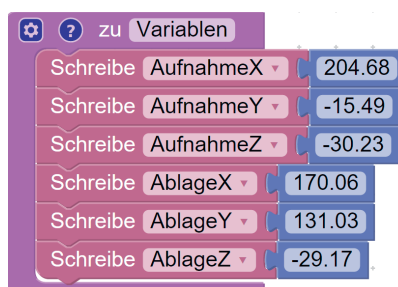


Abbildung 11: Zuweisung aller Koordinaten

Sind nun alle Variablen definiert, so kann mit der Programmierung des Bewegungsablaufs begonnen werden. Hierzu wird zunächst wieder ein Funktionsblock angelegt, welcher mit „Bewegung“ benannt wird. (Der Vorgang wurde in den Abbildungen 2-3 beschrieben).

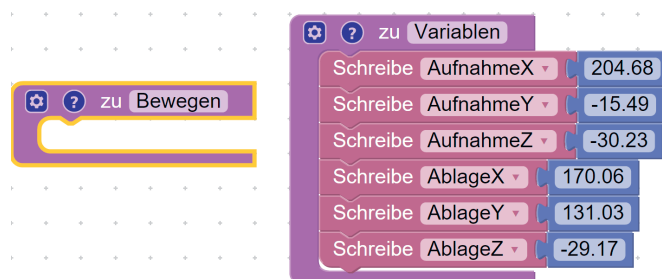


Abbildung 12: Anlegen des Funktionsblocks für die Bewegungsabfolge

Für die Stapel- und Ablagelogik hat sich der Jump-Befehl als vorteilhaft erwiesen, weshalb dieser für die Bewegungsabläufe genutzt wird. Hierzu klickt man in den Programmblöcken auf die DobotAPI und dort auf „Antrag“. Unter Antrag wählt man dann den „Jump To“ Befehl aus und zieht diesen per Drag&Drop in den neuen Funktionsblock „Bewegen“ (siehe Abbildung 13).

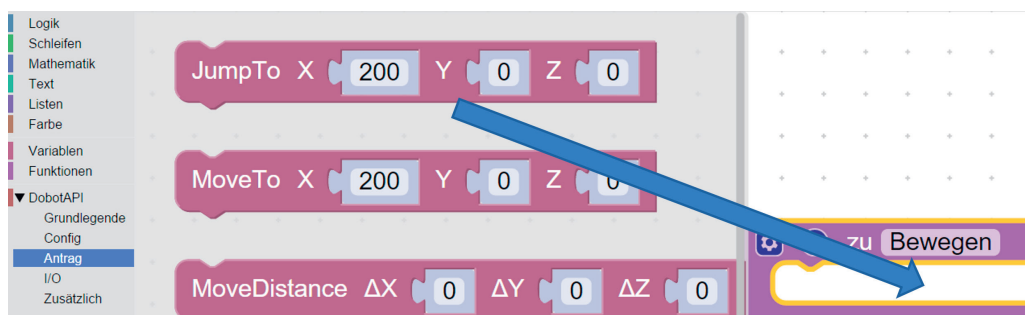


Abbildung 13: Auswahl der Bewegungsart

Jetzt muss man sich Gedanken über die Fahrabfolge machen. Der Dobot soll in diesem Fall zunächst den Aufnahmepunkt A anfahren. Die Koordinaten des Aufnahmepunktes wurden unter dem Funktionsblock „Variablen“ bereits definiert und müssen nun noch auf den Jump To Befehl übertragen werden. Hierzu geht man in den Programmblöcken auf den Reiter Variablen und zieht die entsprechende Variable (hier AufnahmeX, AufnahmeY und AufnahmeZ) in die einzelnen Felder des Jump To Befehls (siehe Abbildung 14-15).

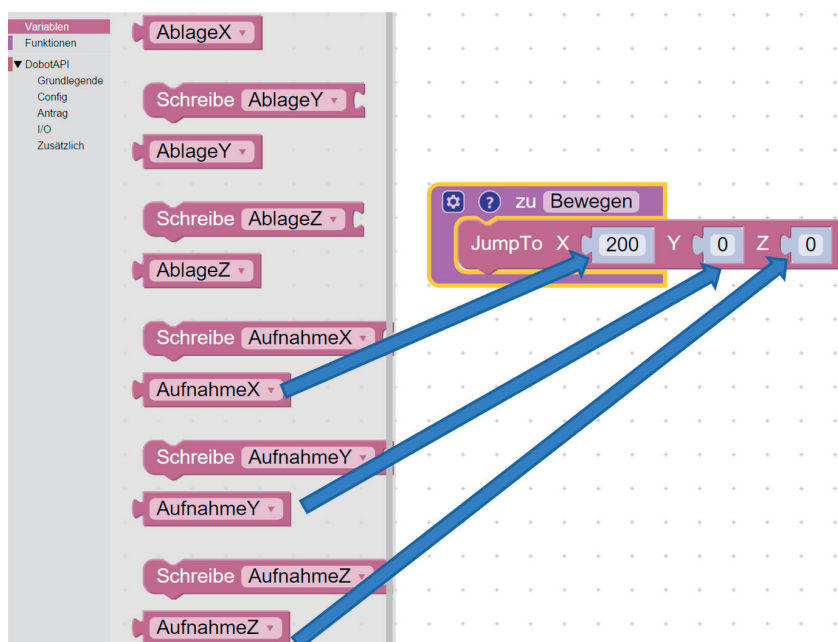


Abbildung 14: Zuweisung Variablen zu Punkten

Sind alle Variablen vollständig hinzugefügt, ergibt sich folgende Ansicht:

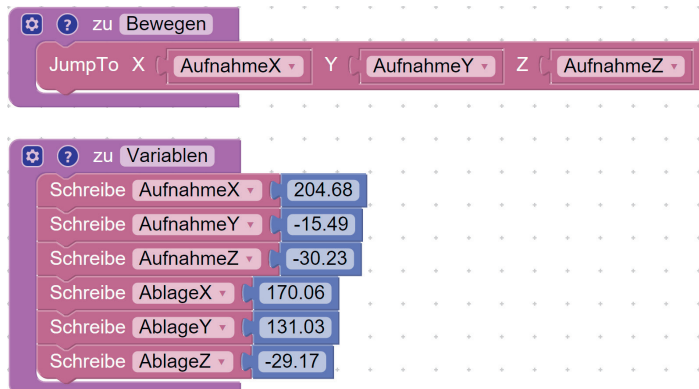


Abbildung 15: Vollständige Variablenzuweisung in einem Jump Befehl

Nun muss die Ablagelogik weiter durchdacht werden. Der Dobot würde sich jetzt an der Aufnahmeposition befinden. Dort soll der Dobot mithilfe des Sauggreifers einen Würfel heranziehen. Hierzu wird in der DobotAPI unter „Antrag“ der Befehl Sauggreifer AN/AUS gewählt und per Drag&Drop unter den Jump To Befehl in unserer Bewegungsfunktion gesetzt (siehe Abbildung 16-17).

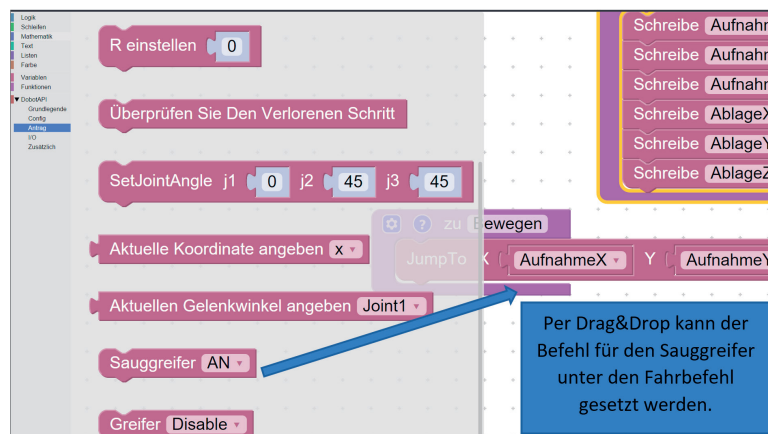


Abbildung 16: Implementation des Saugnapfes

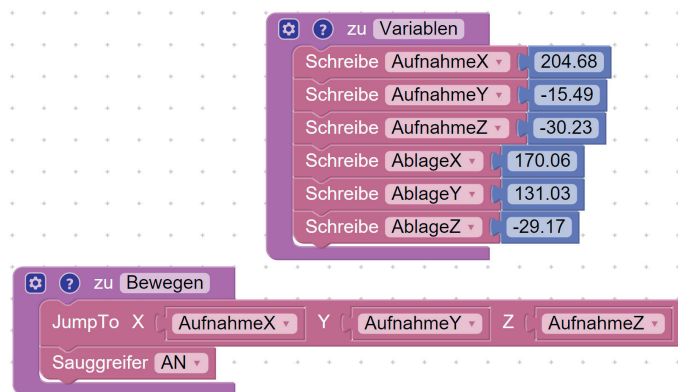


Abbildung 17: Einschalten des Saugnapfes

Nachdem der Sauggreifer eingeschaltet ist, soll der Dobot den Würfel zur Ablageposition bewegen. Hierzu wird wieder ein Jump To Befehl benötigt und unterhalb des Sauggreifer-Befehls im Funktionsblock positioniert. Die Koordinaten werden wieder durch die Variablen definiert, nur, dass dieses Mal die Ablagekoordinaten gewählt werden müssen (AblageX, AblageY und AblageZ). Das Ergebnis ist in Abbildung 18 zu sehen.

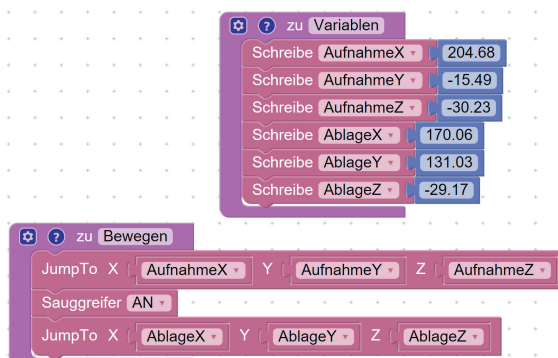


Abbildung 18: Einfügen der Ablageposition

HINWEIS: Wenn man sich viel Arbeit ersparen möchte, kann man den ersten Jump To Befehl (Aufnahme-position) anklicken, kopieren (Strg&C) und über Strg&V wieder einfügen. Hier muss nur beachtet werden, dass die Variablen geändert werden müssen. Hierzu kann das Dropdown Menü geöffnet und die richtigen Variablen ausgewählt werden.

Der Dobot hat den Würfel nun zur Ablageposition befördert, muss diesen allerdings noch loslassen. Hierzu wird erneut der Sauggreifer Befehl implementiert, allerdings wird dieser dieses Mal ausgeschaltet. Damit der Sauggreifer ausgeschaltet werden kann, muss man im Dropdown Menü von „AN“ zu „AUS“ wechseln (siehe Abbildung 19).

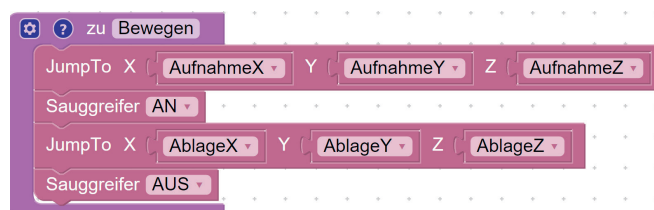


Abbildung 19: Ausschalten des Saugnapfes

Das (fast) fertige Programm für die erste Übung sieht schlussendlich folgendermaßen aus:

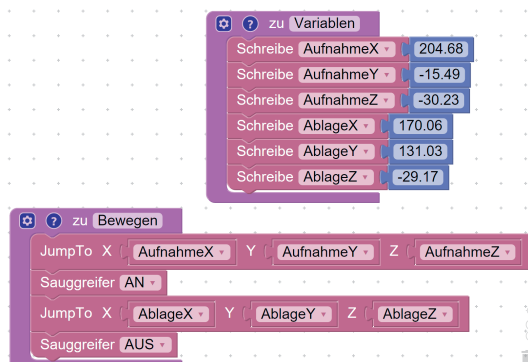


Abbildung 20: (Fast) Fertiger Sketch

Warum fast?

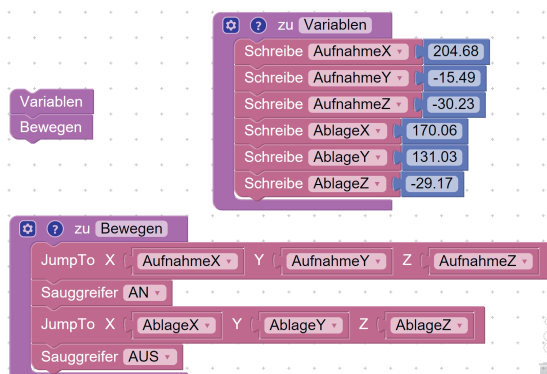


Abbildung 21: Abruf der Unterprogramme

Die Programme wurden hier in Funktionsblöcken definiert. Diese Funktionsblöcke stellen Unterprogramme dar, die noch im Hauptprogramm zu berücksichtigen sind. Damit dies erfolgen kann, gehen Sie in den Programmblöcken wieder auf den Reiter „Funktionen“. Dort erscheinen nun die zwei soeben bearbeiteten Unterprogramme „Variablen“ und „Bewegen“. Diese zieht man per Drag&Drop in die Programmieroberfläche und reiht sie aneinander (siehe Abbildung 21). Nun führt der Dobot (sobald der Start Button betätigt wurde) die beiden Unterprogramme „Variablen“ und „Bewegung“ aus.

### 3.4.2 Ablagelogiken

In der zweiten Übung sollen drei vertikal nebeneinander ausgerichtete Würfel um eine X-Komponente verschoben werden. Hierzu ist eine sogenannte Ablagelogik erforderlich, welche im Folgenden beschrieben wird.

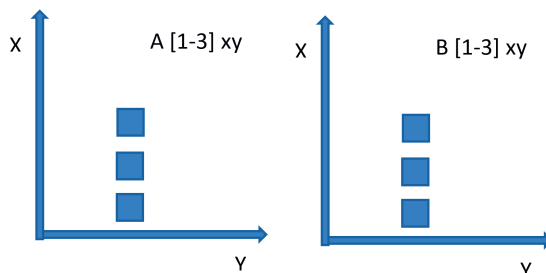


Abbildung 22: Übung zur Ablagelogik

Zunächst ist zu sagen, dass die grundlegenden Variablen aus Übung eins beibehalten werden können. Die Aufnahme- und Ablagepositionen sollen in dieser Übung nicht über das stumpfe Hinzufügen neuer Punkte definiert werden. Vielmehr soll es darum gehen, eine Ablagelogik zu entwickeln, mit der es möglich wird, die Ausgangsvariablen mit jedem Durchlauf zu verändern. Insgesamt sind in dieser Übung drei Würfel zu bewegen, weshalb der Einsatz einer Schleife sinnvoll erscheint. Damit eine Schleife hinzugefügt werden kann, geht man unter den Programmblöcken auf den Reiter „Schleifen“. Wir haben insgesamt drei Würfel, die bewegt werden müssen, weshalb eine Schleife mit n-facher Wiederholung gewählt wird (wie oft wiederholt werden soll, ist individuell anpassbar, indem man den Wert verändert). Wir ziehen die Schleife per Drag&Drop in unseren Funktionsblock aus Übung 1 und ändern den Wert in eine drei, sodass die Schleife drei Mal durchgeführt wird (siehe Abbildung 23).

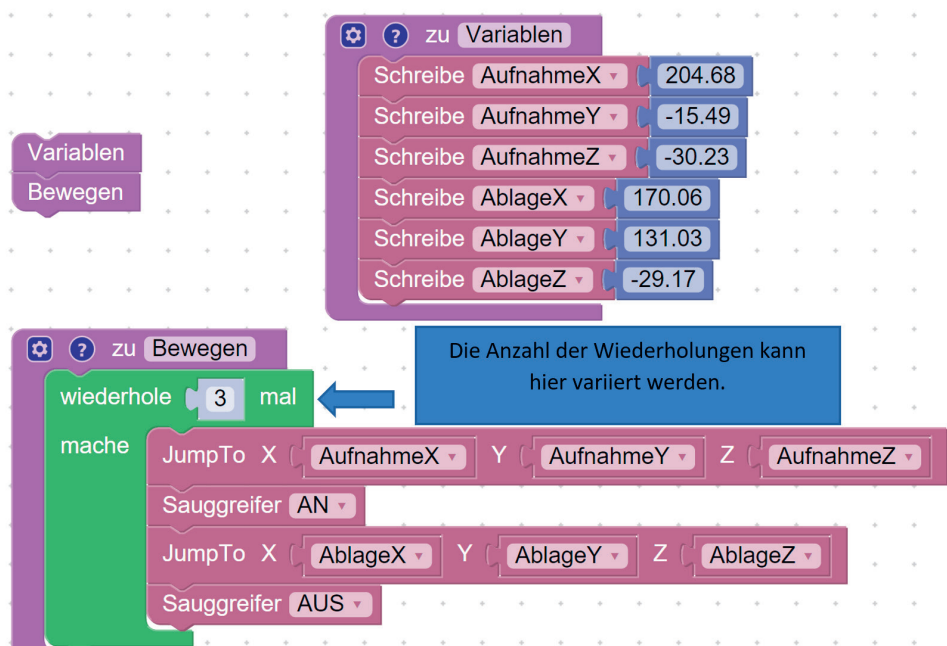


Abbildung 23: Implementation einer Schleife



Würde man den Befehl jetzt ausführen, so würde der Dobot allerdings immer wieder an die gleichen Positionen fahren. Der Dobot soll aber mit jedem Durchgang um einen bestimmten Wert in der X- Achse variieren. Der Startpunkt ist hierbei der rosafarbene Würfel aus der Aufgabenstellung (siehe Abbildung 22). Das bedeutet, dass der Dobot die Aufnahme- und Ablageposition um einen bestimmten X-Wert erhöhen muss, sobald die Schleife durchlaufen ist. Der Verschiebungswert ergibt sich hierbei aus einer gesamten Würfelbreite (wenn dazwischen eine Lücke gelassen wurde, muss die Differenz durch das Anfahren von zwei Positionen bestimmt werden). Damit die Positionen nun neu berechnet werden können, muss im Funktionsblock „Variablen“ eine neue Variable mit dem Namen „VerschiebungX“ angelegt werden, welche mit dem Wert der Verschiebung versehen wird (siehe Abbildung 24).

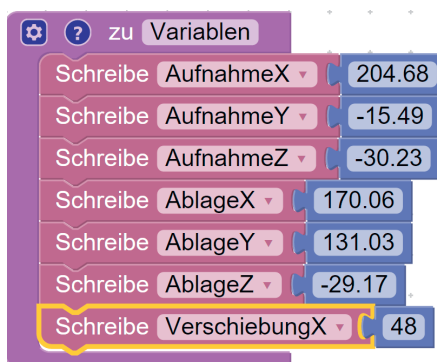


Abbildung 24: Hinzufügen der Verschiebung

Damit der Dobot die Verschiebung nun im Ablauf berücksichtigen kann, müssen die Aufnahme- und Ablagepositionen X innerhalb der Schleife um den Wert der VerschiebungX erweitert werden. Hierzu wird in den Programmblöcken unter dem Reiter „Mathematik“ die Summenfunktion implementiert (siehe Abbildung 25).

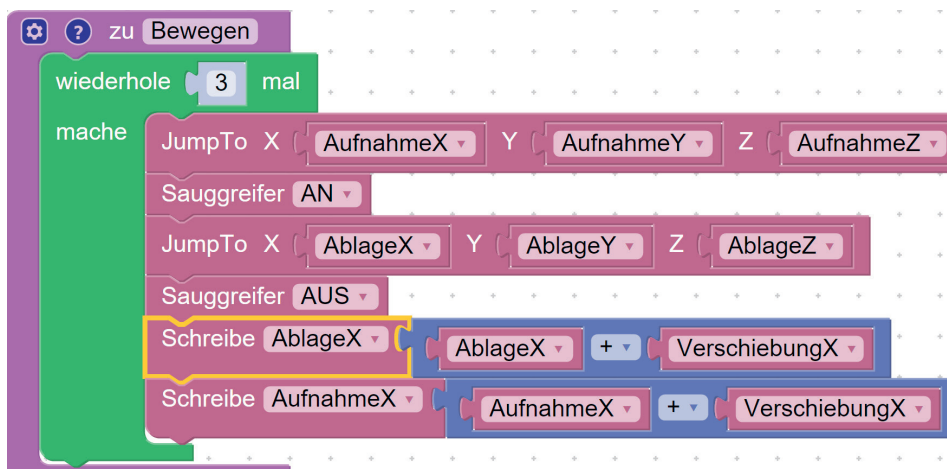


Abbildung 25: Implementierung der Summenfunktion zur Ablage-logik

In einer dritten Übung sollen nun auch wieder drei Würfel bewegt werden. Die drei nebeneinanderliegenden Würfel sollen dieses Mal allerdings aufeinandergestapelt werden (siehe Abbildung 26).

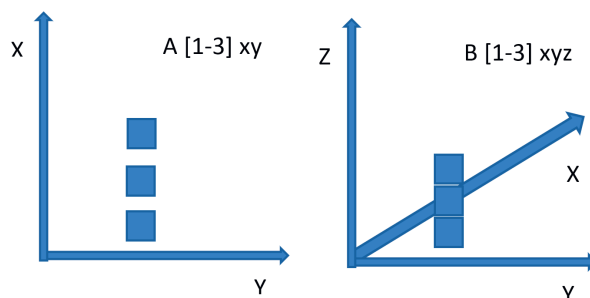


Abbildung 26: Übung 3: Aufeinanderstapeln

Für die Aufnahme der Würfel kann wieder dieselbe Logik verwendet werden wie in Übung zwei, da die Würfel wieder vertikal nebeneinander liegen. Die Ablagelogik hingegen verändert sich. Es ändert sich nicht mehr der AblageX-Wert, sondern die Z-Komponente, da übereinandergestapelt werden soll. Hierzu wird zunächst erneut eine neue Variable angelegt, welche die Höhendifferenz angibt (hierzu entweder die Würfelhöhe messen oder zwei Würfel übereinanderstapeln und die Differenz berechnen). Die Variable wird „VerschiebungZ“ genannt (siehe Abbildung 28). Die Variable „VerschiebungX“ kann bestehen bleiben, da diese für die Aufnahmeposition wieder benötigt wird.

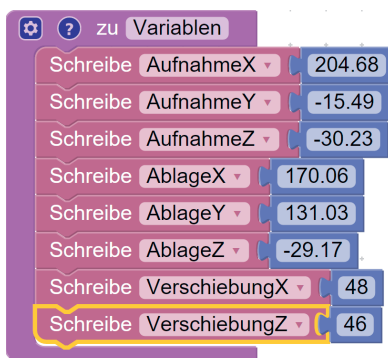


Abbildung 27: Definition der Verschiebung

Anstelle der Änderung der „AblageX“ im Bewegungsablauf verändert sich die „AblageZ“. Hierzu im Dropdown Menü die Elemente verändern, sodass sich folgende Lösung ergibt:

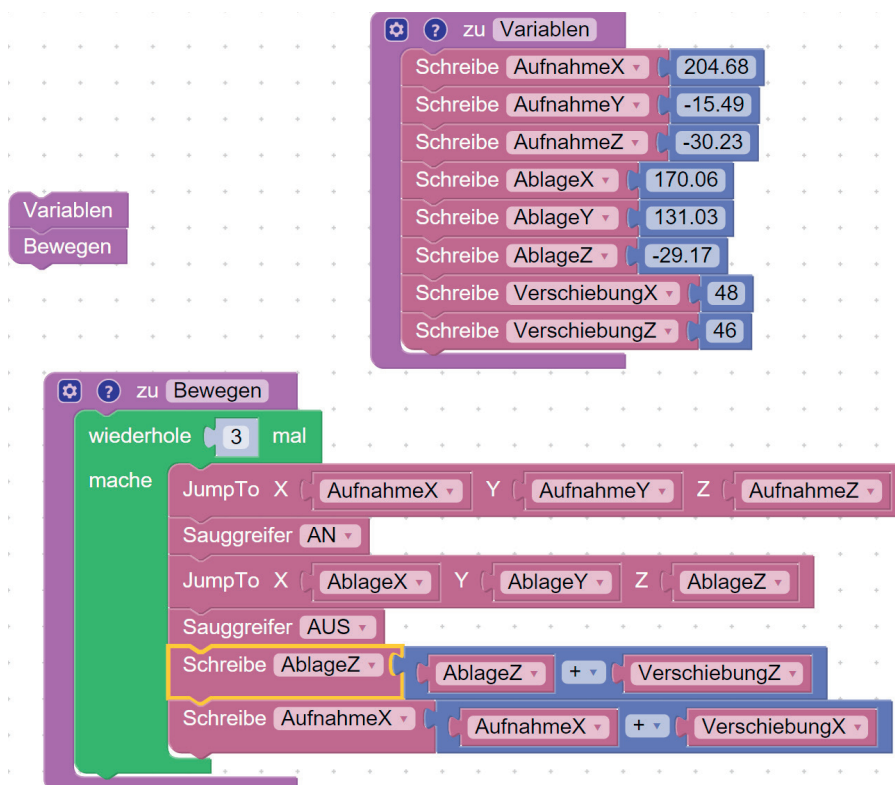


Abbildung 28: Lösung Übung 3

## 3.5 Lichtschranke, Farbsensor und Sortierung

Dani Hamade, Carl von Ossietzky Universität Oldenburg

### 3.5.1 Einbindung des Förderbandes

Die erste Übung besteht hier darin, das Förderband ordnungsgemäß anzuschließen und einen Sketch zu entwickeln, mit dem das Förderband auf Funktion überprüft werden kann. Das Förderband wird hier an den Anschluss „STEPPER1“ angeschlossen. Nun folgt die Funktionsprüfung. Zunächst kann man hierzu eine Funktion anlegen, die den Namen „Funktionsüberprüfung Förderband“ (siehe Abbildung 29) erhält (wie man Funktionen anlegt, kann in Kapitel eins nachgelesen werden).

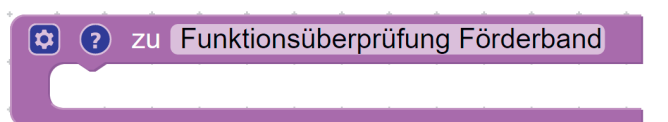


Abbildung 29: Funktion zur Funktionsüberprüfung des Förderbandes

Ein Ansatz, der nun zur Funktionsüberprüfung verfolgt werden kann, ist über eine Endlosschleife. Hierzu nimmt man den Block „wiederhole solange“ aus Blockly (DOBOTSTUDIO) unter „Schleifen“ und setzt ihn unter die Funktion (siehe Abbildung 30).

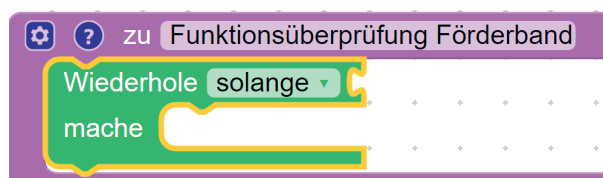


Abbildung 30: Implementation der Schleife in die Funktion

Der Ausdruck „wiederhole solange“ (in Python „while“) muss, damit daraus eine Endlosschleife wird, an eine Bedingung geknüpft werden. Hier kommt der boolesche Ausdruck „wahr“ (in Python „true“) zum Einsatz (bei Blockly zu finden unter „Logik“). Verknüpft man diese beiden Bestandteile, so erhält man eine Endlosschleife (siehe Abbildung 31).

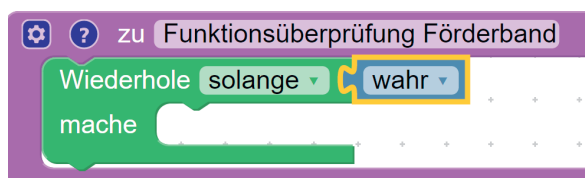


Abbildung 31: Implementation Endlosschleife

Der nächste Schritt ist, dass das Förderband eingebettet wird. Mit dem hier gewählten Ansatz wird das Förderband permanent laufen, sodass man die Funktionsweise überprüfen kann. Die Befehle für das Förderband (und auch andere Peripherie) sind in der DOBOTAPI unter „Zusätzlich“ zu finden. Wichtig ist, dass der richtige Anschluss für den Schrittmotor des Förderbandes im Auswahlménü des Befehls gewählt wird. Im Fall von Abbildung 32 ist der Schrittmotor an den Anschluss „Stepper 1“ des DOBOTs angeschlossen (siehe Abbildung 32). Zusätzlich dazu lässt sich auch die Geschwindigkeit des Schrittmotors einstellen (Einheit mm/s). Zu beachten ist hierbei, dass die maximale Geschwindigkeit mit 120 mm/s angegeben ist. Die Erfahrung hat gezeigt, dass es ab 100 mm/s bereits zu Problemen kommen kann. Zum Schluss muss die Funktion nur noch ausgeführt werden (siehe Abbildung 33):

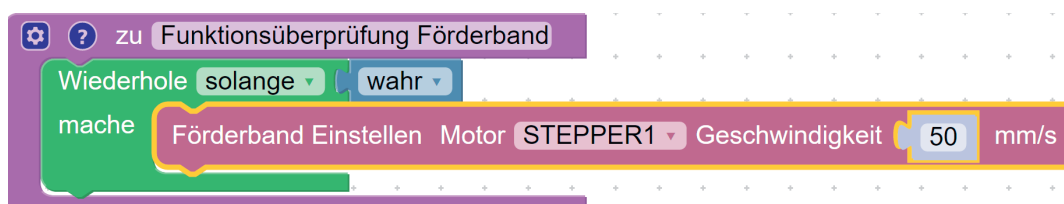


Abbildung 32: Einbettung des Förderbandes in die Endlosschleife

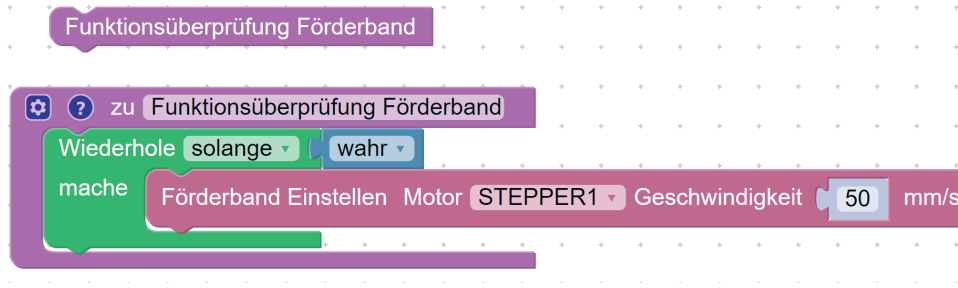


Abbildung 33: Fertiger Sketch zu Übung 1a

In einer weiteren Übung soll nun ein Sketch entwickelt werden, bei dem das Förderband in einem Rhythmus von fünf Sekunden ein- und ausgeschaltet wird. Zu beachten ist hierbei, dass dies in einer Dauerschleife stattfindet. Der Ansatz zur Umsetzung einer Dauerschleife wurde in der Lösung zur ersten Übung mit dem Förderband bereits dargelegt und kann auf diese Übung übertragen werden (siehe Abbildung 34).

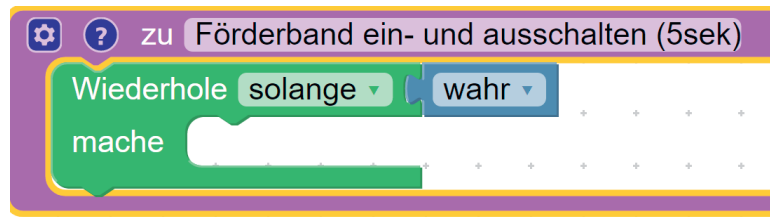


Abbildung 34: Dauerschleife für den Rythmus aus Übung 1b

Wie das Förderband angesprochen werden kann, ist aus der ersten Übung abzuleiten. Dieses Mal muss aber zusätzlich dazu eine Verzögerungszeit implementiert werden. Damit der Förderprozess pausiert werden kann, ist die Geschwindigkeit auf 0 mm/s einzustellen (siehe Abbildung 35).

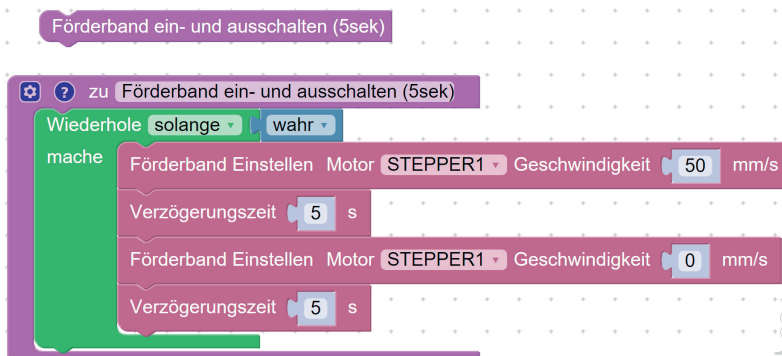


Abbildung 35: Lösung Übung 1b

Nun soll der Dobot Würfel auf dem Förderband ablegen, welche dann vom Förderband abtransportiert werden. Hierzu sind zunächst die Positionen für die Aufnahme und die Ablage als Variablen zu definieren (siehe Abbildung 36). Zusätzlich dazu, muss für diese Übung eine Aufnahmelogik eingebettet werden, weshalb es sinnvoll ist, zusätzlich die Breite eines Würfels zu definieren (oder die Würfelhöhe, je nach Aufnahmelogik). Falls die Vorgehensweise nicht mehr bekannt ist, so kann in Kapitel eins nachgeschaut werden.

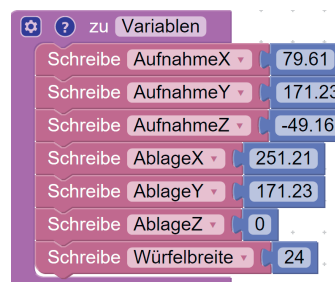


Abbildung 36: Positionsdefinitionen (individuell anpassen!)

Eine Endlosschleife wie in den Übungen zuvor würde bei der Aufnahmelogik zu Komplikationen führen, da der Arbeitsbereich des DOBOTs an seine Grenzen stoßen würde. Aus diesem Grund ist eine Schleife mit vorab definierter Wiederholungsanzahl zu wählen (in diesem Fall sollen drei Durchläufe stattfinden). Mit Einbettung der Auf- und Abnahme der Würfel sowie der Ablagelogik ergibt sich der in Abbildung 37 abgebildete Programmaufbau (siehe Abbildung 37). Zunächst fährt der DOBOT zu der Aufnahmeposition mittels JUMP-Befehl und schaltet dort den Sauggreifer ein, um einen Würfel aufzunehmen. Anschließend springt der DOBOT mit angesaugtem Würfel auf das Förderband. Damit es beim Anfahren des Förderbandes zu keiner Kollision mit dem Saugnapf und dem Würfel kommt, fährt der DOBOT um 10mm nach oben ( $\Delta z=10$ ). Der Würfel ist abgelegt und das Förderband kann losfahren, was mit dem nächsten Befehl geschieht (die gewählte Geschwindigkeit beträgt hier 50mm/s). Das Förderband fährt hier insgesamt fünf Sekunden und hält dann wieder an. Als nächstes ist nur noch die Aufnahmelogik zu implementieren. In diesem Fall sind die Würfel horizontal nebeneinander gelagert, sodass die Y-Koordinate der Aufnahmeposition um eine Würfelbreite erweitert werden muss. Nach Durchführung der drei Durchläufe sind dementsprechend alle drei nebeneinander gelegten Würfel nacheinander abtransportiert (hier ein Video zum Programmablauf: <https://youtube.com/shorts/74nZLB-n04U?feature=share>).

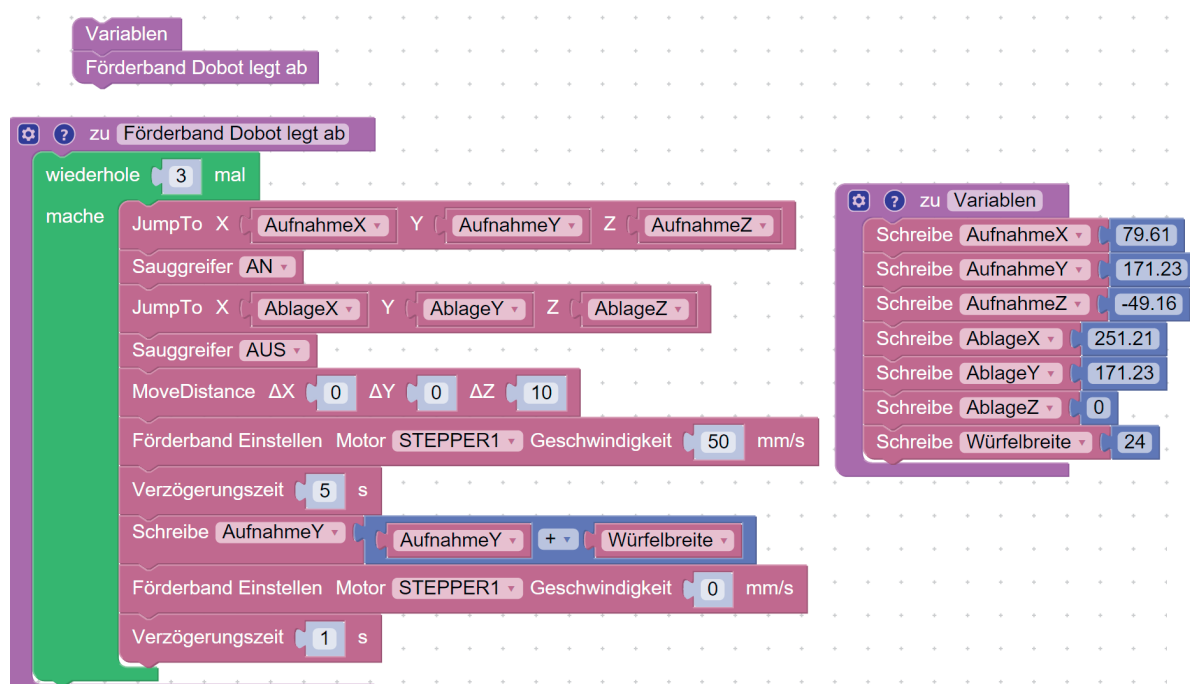


Abbildung 37: Lösung für die Übung 1c mit Aufnahmelogik

### 3.5.2 Einbindung der Lichtschanke

In diesem Abschnitt geht es um den ordnungsgemäßen Anschluss der Lichtschanke. In den seitlichen Schienen des Förderbandes befinden sich Gewindeplatten mit einem M6 Innengewinde (siehe Abbildung 38).

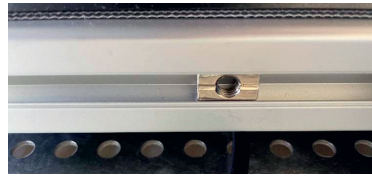


Abbildung 38: Verstellbare Gewindeplatte am Förderband



Abbildung 39: Befestigung der Lichtschanke an der Gewindeplatte des Förderbandes



Abbildung 40: Anschluss Lichtschanke DOBOT

An diesen kann die Lichtschanke mit einer M6 Schraube befestigt werden (siehe Abbildung 39). Nach ordnungsgemäßer Befestigung kann die Lichtschanke an den DOBOT angeschlossen werden (richtigen Port beachten! Siehe Abbildung 40).

In der Übung soll die Lichtschanke nun auf Funktion überprüft werden. Hierzu soll eine Printausgabe bei Blockly erstellt werden, aus der hervorgeht, ob ein Objekt erkannt wird oder nicht. Ein Ansatz ist hier, dass wieder eine Funktion mit dem Titel „Printausgabe Lichtschanke“ erstellt wird. In dieser kann die Lichtschanke zur besseren Übersicht zunächst als Variable definiert werden. Die entsprechenden Befehle für die Lichtschanke sind in der DOBOT API unter „Zusätzlich“ zu finden. Die Funktion soll fortlaufend überprüft werden, weshalb wieder eine Endlosschleife gesetzt wird (siehe Abbildung 41).

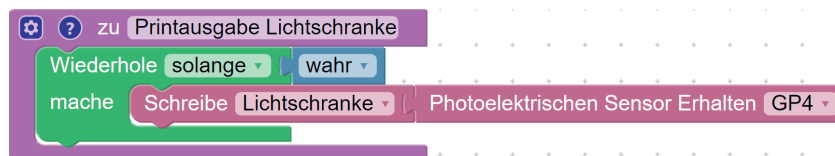


Abbildung 41: Definition der Lichtschanke als Variable (Auswahl des richtigen Ports beachten!)

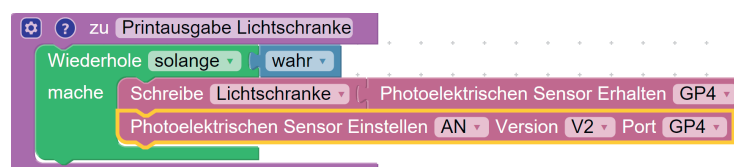


Abbildung 42: Einschalten der Lichtschanke

Nun muss die Lichtschranke initiiert werden, was durch das Einschalten dieser innerhalb des Sketches geschieht. Hierzu ist der Befehl wie in Abbildung 42 zu ergänzen. Auch hier ist wieder auf die richtige Version und die korrekte Auswahl des Ports zu achten. Das Ziel ist es nun, eine Printausgabe zu erhalten, WENN ein Objekt erkannt wird. ANSONSTEN soll eine Printausgabe erfolgen, aus der hervorgeht, dass kein Objekt erkannt wird.

Hier bietet sich, wie aus der Betonung der Bedingungen schon hervorgeht, eine WENN DANN SONST Bedingung an. Diese ist bei Blockly unter „LOGIK“ zu finden. Die SONST Bedingung lässt sich über das Zahnrad hinzufügen (siehe Abbildung 43).

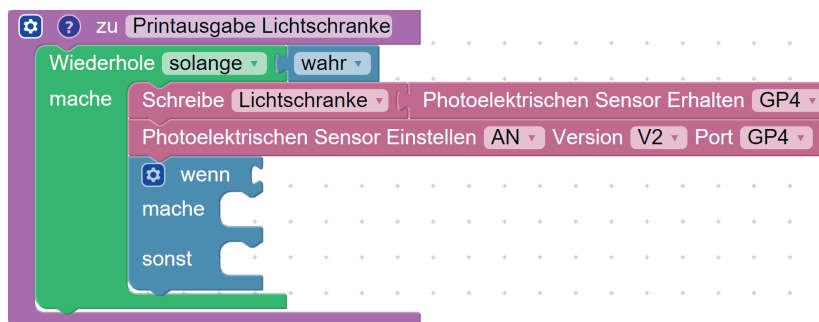


Abbildung 43: Implementation der Bedingungen

Damit die Printausgabe erfolgen kann, sind nun nur noch die Bedingungen zu definieren. Die Lichtschranke gibt einen Integer-Wert zurück, weshalb die Bedingung sein muss, dass WENN ein Objekt erkannt wird, die Variable Lichtschranke gleich eins sein muss. Über den Reiter „Text“ lässt sich dann eine Ausgabe implementieren, in die geschrieben werden kann „Objekt erkannt“ (siehe Abbildung 44). Wird kein Objekt erkannt, so soll in diesem Beispiel folgende Ausgabe erfolgen: „kein Objekt“. Der fertige Sketch zur Funktionsüberprüfung nimmt somit folgende Gestalt an:

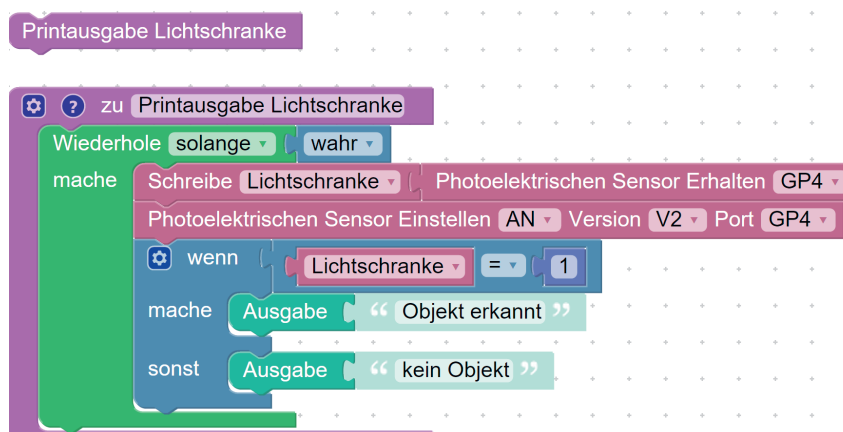


Abbildung 44: Fertiger Sketch zur Funktionsüberprüfung der Lichtschranke

Die dazugehörige Printausgabe erscheint nach dem Starten des Programmes im Protokoll auf der rechten Seite (siehe Abbildung 45).

```
Protokoll:
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
[16:24:34]Kein Objekt
```

Abbildung 45: Protokoll zur Printausgabe der Objekterkennung

Nun, da die Lichtschranke funktioniert soll eine weitere Übung folgen, in welcher auch das Förderband eingebunden wird. In dieser Übung soll der zuvor erstellte Sketch angewendet werden, sodass das Förderband für zwei Sekunden stoppt, sobald ein Würfel erkannt wurde und diesen nach Ablauf der zwei Sekunden abtransportiert. Der Grundaufbau aus der Übung zur Lichtschranke kann übernommen werden und um die Befehle zur Ansteuerung des Förderbandes ergänzt werden. Der hier verfolgte Ansatz (siehe Abbildung 46) ist, dass die Geschwindigkeit des Förderbandes für zwei Sekunden auf 0 mm/s gesetzt wird, sobald ein Objekt erkannt wird. Anschließend soll das Förderband mit gewohnter Geschwindigkeit von 50 mm/s weiterfahren. Wird kein Objekt erkannt (also SONST), so soll das Förderband einfach permanent mit 50mm/s fahren. Wichtig ist hierbei allerdings, dass hier ebenfalls eine Verzögerungszeit von zwei Sekunden angegeben wird, da das Förderband sonst „stottern“ würde, weil der Würfel über die gesamte Breite immer wieder neu als Objekt erkannt wird.

Eine weitere Möglichkeit das „Stottern“ zu umgehen ist, dass eine zusätzliche Schleife eingebunden wird, in der die Aussage getroffen wird, dass „nichts“ gemacht werden soll, solange ein Objekt erkannt wird. Auf diese Weise fährt das Förderband nicht permanent weiter, während der Würfel von der Lichtschranke erkannt wird (siehe Abbildung 19).

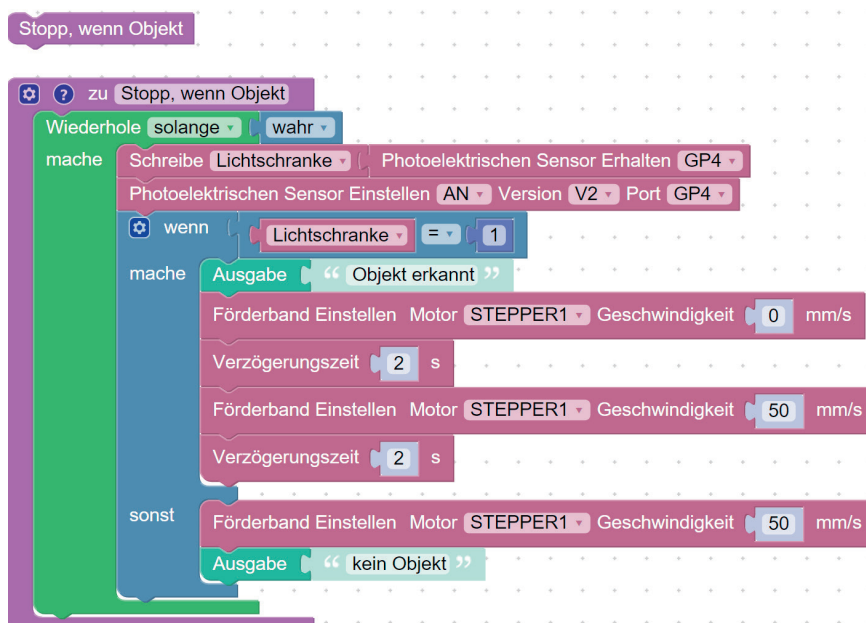


Abbildung 46: Sketch zu Übung 2c, Lösungsweg 1

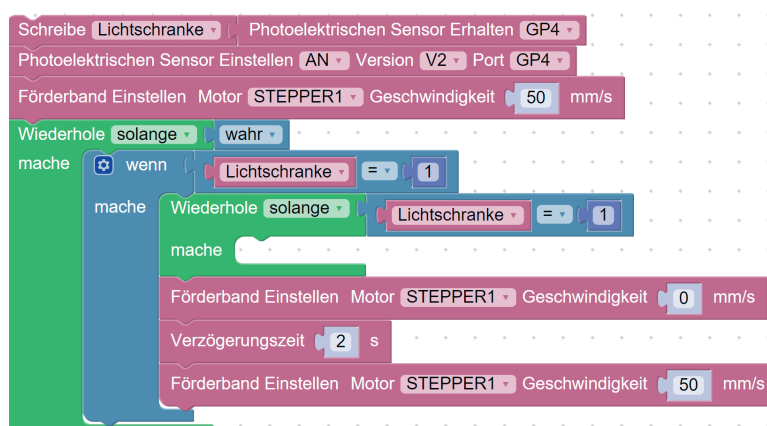


Abbildung 47: Sketch zu Übung 2c, Lösungsweg 2

Aufbauend auf der Übung soll nun der DOBOT eingebettet werden. Dieser soll nämlich nach der Objekterkennung (Würfel) eingreifen und das erkannte Objekt in einer Box lagern. Hierzu werden wieder sowohl Aufnahme- als auch Ablageposition als Variablen definiert. WENN die Lichtschranke nun ein Objekt erkennt, dann soll der DOBOT zur Aufnahme-Position fahren und den Sauggreifer einschalten. Nachdem der Würfel aufgenommen wurde, fährt der DOBOT zur Ablage-Position und schaltet den Sauggreifer darüber ab. Wird kein Würfel erkannt (also SONST), so kann das Förderband weiter fördern. Es ergibt sich der in Abbildung 48 abgebildete Sketch.



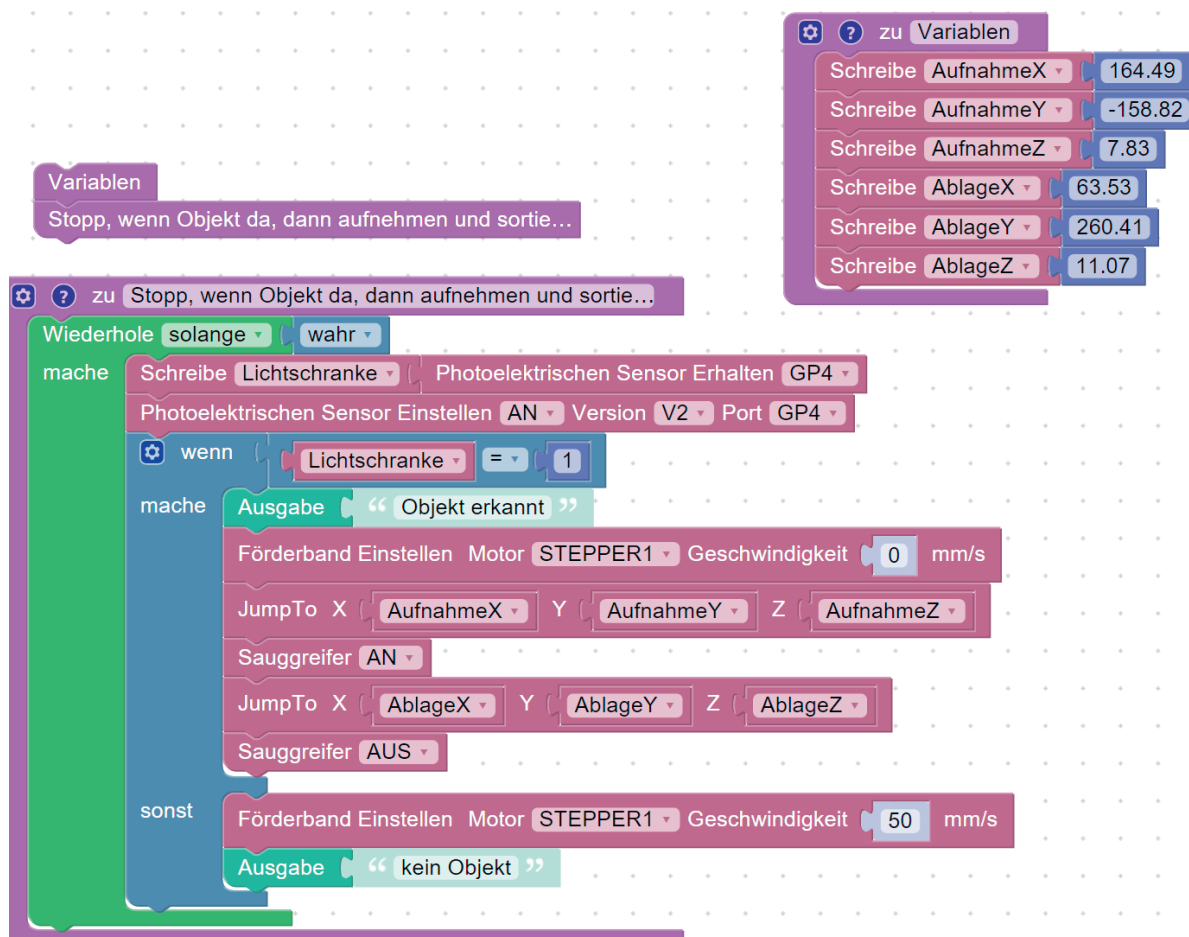


Abbildung 48: Fertiger Sketch zu Übung 3

### 3.5.3 Farbsensor

3.5

Farbsensor

Nach erfolgreichem Aufbau des Farbsensors (Port GP2), soll in dieser Übung ein Sketch entwickelt werden, bei dem die vom Farbsensor erkannten Farben in einer Printausgabe angezeigt werden. Wie eine Printausgabe erstellt werden kann, ist aus den vorherigen Übungen zu entnehmen. Was in dieser Übung neu hinzukommt, sind die Befehle zur Ansteuerung des Farbsensors. Diese sind in der DOBOT-API unter „Zusätzlich“ zu finden (siehe Abbildung 49).

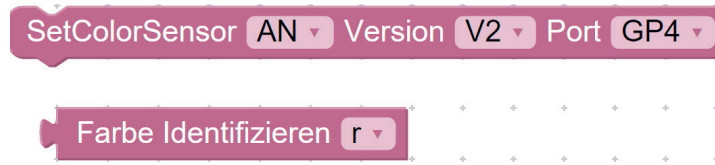


Abbildung 49: Befehle zur Steuerung des Farbsensors

Man beachte, dass der Farbsensor zunächst aktiviert, die richtige Version ausgewählt und der Port definiert werden muss (analog zur Lichtschranke). Der Befehl „Farbe identifizieren“ ist der für das Programm ausschlaggebende, da hier der Abruf zur Farberkennung gestartet wird. Die verschiedenen Farben (RGB) können über das Auswahlménü variiert werden (siehe Abbildung 49). Damit die Farberkennung nun gestartet werden kann, ist wieder eine Endlosschleife zu implementieren, sodass die Farberkennung dauerhaft durchläuft. Anschließend müssen die Farbwerte abgefragt werden (Integer) und eine Printausgabe gesetzt werden, WENN eine Farbe erkannt wurde (siehe Abbildung 50).

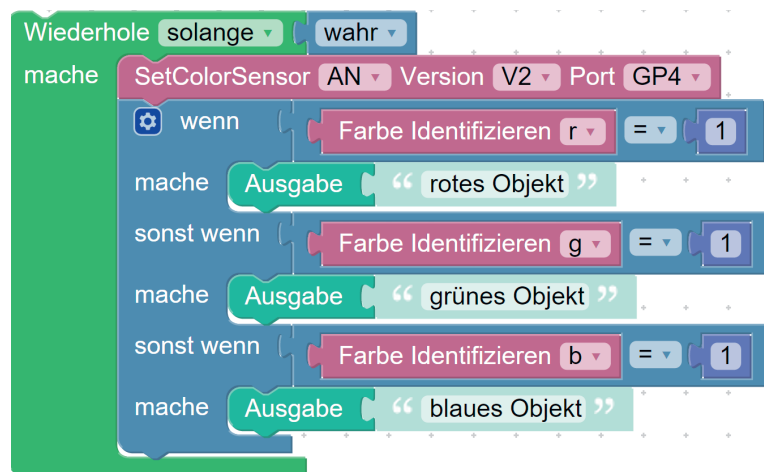


Abbildung 50: Printausgabe Farberkennung

Hat die Printausgabe funktioniert, kann nun eine Sortierung stattfinden. In dem Programm sollen Förderband, Lichtschranke, DOBOT und Farbsensor zusammenwirken. Zunächst sollte das Förderband stoppen, wenn ein Objekt erkannt wurde. Dieses Objekt sollte vom DOBOT aufgenommen und über den Farbsensor gelegt werden. Anschließend soll im Programm eine Farberkennung stattfinden, sodass der DOBOT die Würfel entsprechend der Farbe sortieren kann. Hierbei sollten alle roten Würfel in einer gesonderten Box und alle andersfarbigen in einer gemeinsamen Box gelagert werden.

In dem hier verfolgten Ansatz werden zunächst unter der Funktion „Variablen“ wieder alle Positionen definiert (siehe Abbildung 51). Zusätzlich dazu wird dem „Farbe identifizieren“ Befehl eine jeweilige Farbe zugeschrieben. In der Funktion zur Farbsortierung werden dann innerhalb einer Endlosschleife die Befehle so zusammengeführt, dass eine Farberkennung erfolgen kann.

Zunächst muss aber das Förderband gestoppt werden, sobald ein Würfel erkannt wurde. Anschließend wird der Würfel aufgenommen und über den Farbsensor gefahren. Je nach Farbe wird dann entsprechend sortiert.

Möchte man nun, dass jede Farbe eine Ablagebox erhält, muss lediglich eine neue Position zu den Variablen hinzugefügt werden. Es bietet sich an, die Position BOXGB (also Box grün blau) umzubenennen in BOXG (also Box für grüne Würfel). Anschließend kann eine neue Position für die blauen Würfel, also BOXB hinzugefügt werden (eine Position beinhaltet immer drei Variablen für die Koordinaten X;Y;Z). Zusätzlich zu den Positionsdefinitionen müssen die Positionen innerhalb der Funktion zur Farbsortierung angepasst werden (siehe Abbildung 52).

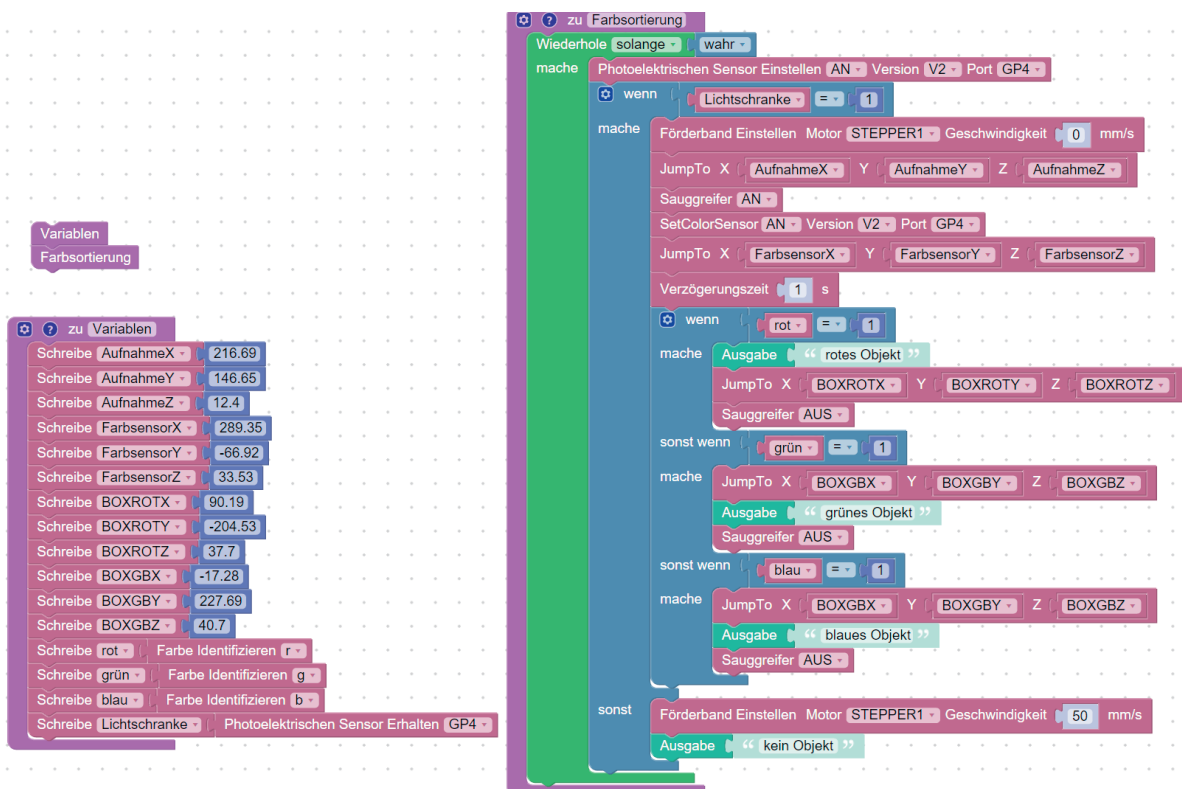


Abbildung 51: Lösung zu Übung 5

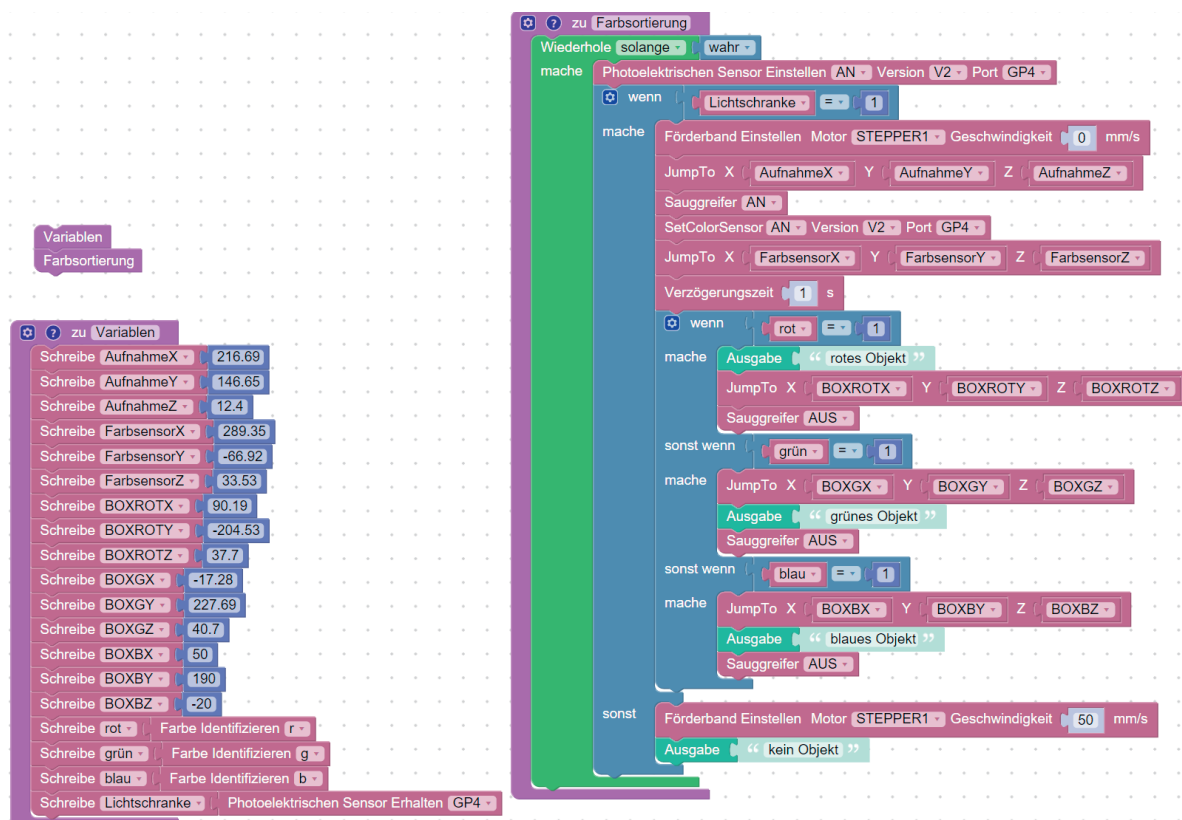


Abbildung 52: Lösung zu Übung 6

## 3.6 Linearachse

Dani Hamade, Carl von Ossietzky Universität Oldenburg

Die Linearachse ist eine Komponente, welche häufig in der Industrie, z. B. zur Bestückung von Maschinen mit Bauteilen, in Lagerwirtschaftssystemen, Produktionslinien etc., eingesetzt wird. Der Grund dafür ist, dass die Linearachse den Arbeitsbereich erheblich vergrößern kann und das Verfahren großer Differenzen dadurch mit hoher Geschwindigkeit automatisiert werden können, welches wiederum in flexibleren Produktionslösungen resultiert. Zu den Nachteilen zählt das Gewicht, da es zumeist zu schweren Konstruktionen führt und zusätzlich erhöht jede ergänzte Komponente die Fehleranfälligkeit, sodass auch die Linearachse für Störungen verantwortlich sein kann.

### 3.6.1 Verbindung mit der Linearachse

Bei Verwendung der Linearachse sind jeweils eigene Anschlüsse zu verwenden, welche sich unterhalb der Platte befinden, auf welcher der Dobot montiert werden muss. So muss der gelbe Stecker in die „Stepper2“-Buchse und der grüne in „GP2“, welche zusammen mit der eigenen Stromversorgung ebenfalls aus der Schleppkette jeweils am Sockel des Dobots angeschlossen werden müssen. Die Linearachse selbst benötigt eine Stromversorgung mit einem Netzteil am äußeren Ende der Linearachse, von welchem ebenfalls ein USB-Kabel für den Rechner verläuft. Die Auswahl der Linearachse erfolgt oben im Fenster von DobotStudio, wie auch schon beim Sauggreifer, wo es nach Auswahl blau angezeigt wird.

### 3.6.2 Steuerung der Linearachse

Die Linearachse wird mit dem Block *SetLinearRail*, welcher in Abbildung 53 dargestellt ist, initialisiert.

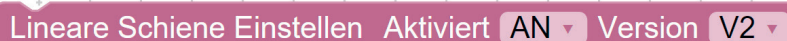


Abbildung 53: Initialisierung der Linearachse

Die Definition von Geschwindigkeit und Beschleunigung findet mithilfe des Blockes *SetLinearRailSpeed* (siehe Abbildung 54) statt. Die höchstmögliche Geschwindigkeit der Linearachse liegt bei 150 mm/s und die Beschleunigung bei 150 mm/s<sup>2</sup>.



Abbildung 54: Einstellen der Geschwindigkeit und Beschleunigung

Um zu bestimmen, zu welcher Position der Dobot entlang der Linearachse fahren soll, wird der Block *MoveLinearRailTo* verwendet (siehe Abbildung 55). Diese Koordinate beschreibt die absolute Position des Dobots auf der Linearachse in Millimeter, sodass der Dobot sich nicht bewegen wird, falls er bereits an der definierten Stelle steht. Der Bewegungsbereich (in mm) erstreckt sich von 0 bis 1000. Zusätzlich kann die Steuerung des Dobots auf der Linearachse manuell im Operation-Panel (via L+ und L-) erfolgen.

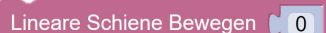


Abbildung 55: Befehl zur Bewegung der Linearachse

Im Fall, dass diese Buttons im Operation-Panel nicht dargestellt werden, muss unter dem Liste-Symbol rechts in der Ecke des Operation Panels ein Haken unter „Linear Rail control“ gesetzt werden. Wie schon zuvor, ist ein Homing des Dobots sofort nach Verbindung mit der Linearachse durchzuführen, bei welchem der Dobot sowohl Start- als auch Endposition anfährt, damit keine Koordinationsschwierigkeiten auftreten. Ein Programm zum Abfahren des Start- und Endpunktes könnte z. B. so aussehen, wie in Abbildung 56 dargestellt.

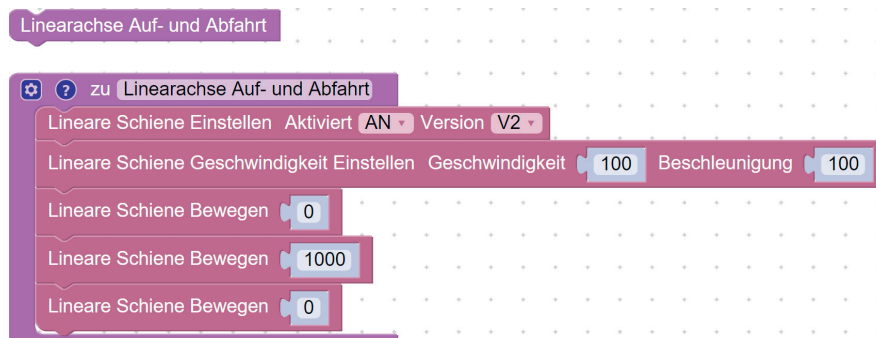


Abbildung 56: Beispiel zum Abfahren von Start- und Endposition

### 3.6.3 Interaktion mit dem Dobot und der Linearachse

Die Linearachse lässt sich nun mit dem Dobot problemlos ansteuern, mit dem Vorteil, dass sich die Bewegung um einen Freiheitsgrad und damit um eine Variable erhöht hat. Falls nun zum Beispiel neun Würfel (3 x 3) um einen Meter versetzt werden sollen (siehe Abbildung 57), lässt sich dies auf simple Weise realisieren.

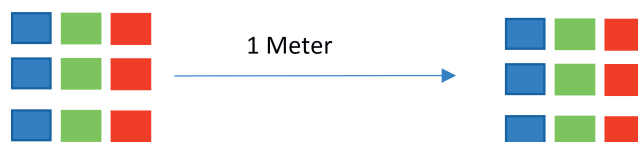


Abbildung 57: Übung Würfelmatrix mit Linearachse

Durch die Bewegung der Linearachse ist die Entfernung von 1 m bereits gewährleistet, wenn die *MoveLinearRail*-Blöcke auf 0 und 1000 jeweils gesetzt werden. Eine Änderung der Dobot-Positionskoordinaten ist damit nicht zwangsläufig notwendig. Eine Stellung, in die der Dobot zurückkehrt, wenn er sowohl aufgenommen als auch abgelegt hat, ist sinnvoll, um bei der Fahrt über die Linearachse eine Kollision mit Hindernissen, wie z. B. anderen Würfeln, zu verhindern. Außerdem findet die Initialisierung der Würfelbreite statt, deren Bedeutung im Folgenden näher erklärt wird. Die Variablen-Funktion kann folgendermaßen aussehen (siehe Abbildung 58).

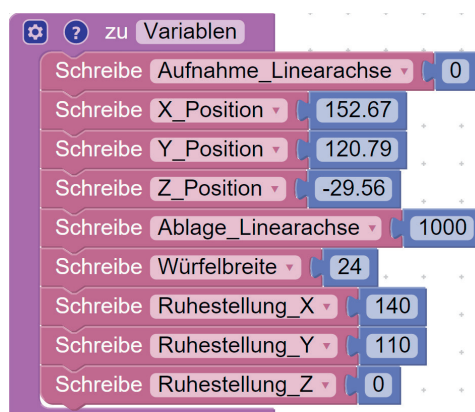


Abbildung 58: Funktion der Variablen

Nun können die zwei Aufgaben des Dobots (Aufnahme und Ablage) definiert werden. Die X-, Y- und Z-Koordinaten sind bei beiden gleich, während sich die Linearachsen-Koordinate und die Sauggreifer-Funktion (on/off) ändern. Die Abbildungen 59 und 60 stellen die beiden Prozesse jeweils dar. Bei der Z-Position wird ein höherer Wert angegeben als eigentlich zutreffend und dann via *MoveDistance* angenähert, um den Würfel von oben abzusetzen und zu verhindern, dass bereits platzierte Würfel verschoben werden.

Ein direkter *MoveTo*-Befehl hätte dagegen eine Bewegung in gerader Linie zum Bestimmungsort veranlasst. In dem eigentlichen Skript können diese Komponenten zusammengefasst werden. Ein direkter *MoveTo*-Befehl hätte dagegen eine Bewegung in gerader Linie zum Bestimmungsort veranlasst. In dem eigentlichen Skript können diese Komponenten zusammengefasst werden.

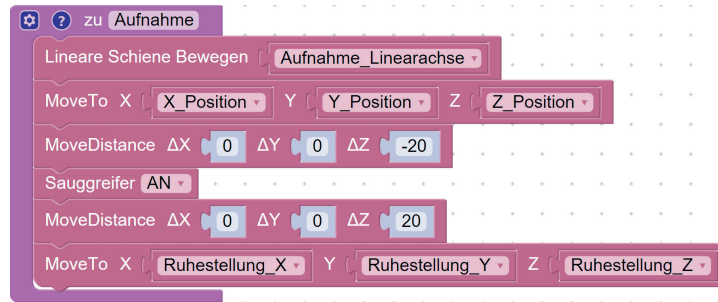


Abbildung 59: Funktion der Aufnahmeposition und der Bewegung

Mithilfe des Variablen-Aufrufs wird, wie zuvor, sichergestellt, dass die Variablen zugreifbar sind, woraufhin die Linearachse, zusätzlich zur Geschwindigkeit und Beschleunigung, initialisiert wird. Mit dem Aufbau, wie in Abbildung 5 zu sehen, und der hier getätigten Programmstruktur müssen die Variablen jeweils neu angepasst werden. In der Skizze stehen jeweils drei Würfel nebeneinander, d. h. man kann eine Schleife erstellen, die dreimal eine Handlung ausführt und nach jedem Schleifendurchlauf die X-Positionskoordinate um den Würfelbreite-Wert ändert. Daraufhin sollte in diesem Fall erneut eine Koordinate geändert werden, die Y-Koordinate.

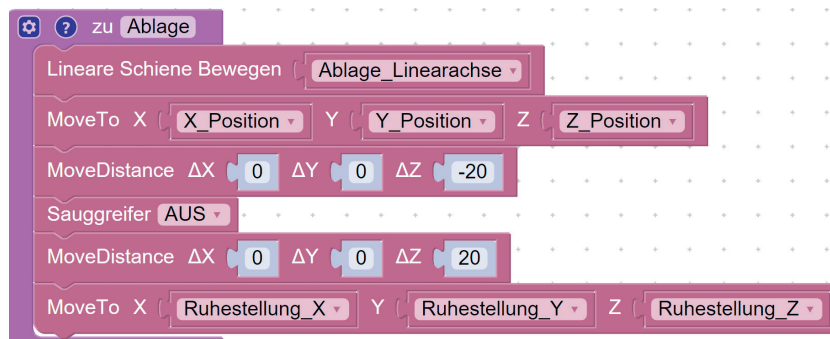


Abbildung 60: Funktion der Ablageposition und Bewegung

Bei direkt aneinander liegenden Würfeln wäre der nächste Würfel um eine Würfelbreite in Y-Richtung entfernt. Allerdings würde die X-Positionskoordinate lediglich für einen weiteren Schleifendurchlauf stimmen, da sie weiterhin um die Würfelbreite erhöht wird. Zur Verhinderung dessen wird die X-Koordinate also wieder auf den Initialwert gesetzt.

Der gesamte Prozess soll dreimal durchgeführt werden, weshalb sich wieder eine repeat-Schleife anbietet. Das Resultat des beschriebenen Vorgangs zum (Haupt-)Skript ist in Abbildung 61 dargestellt.

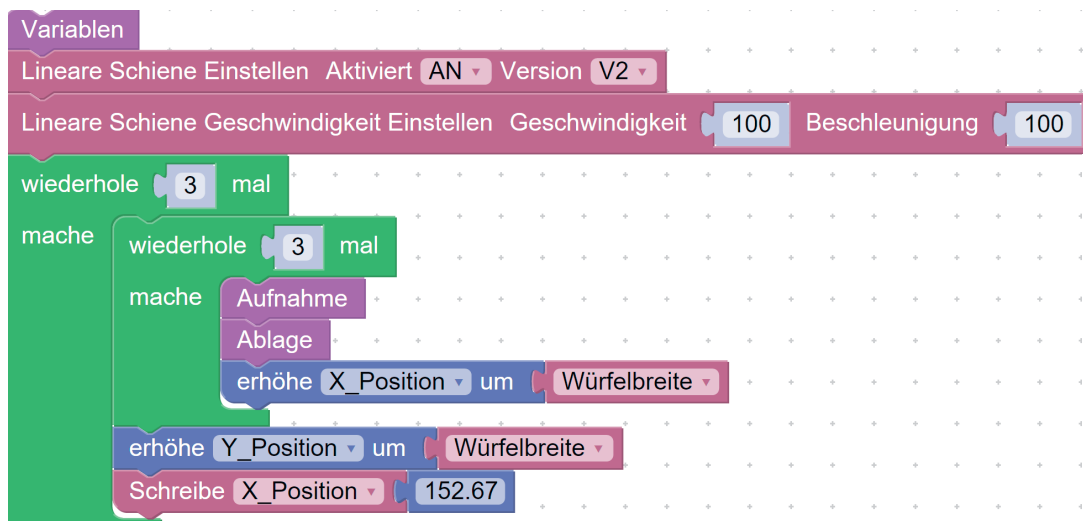


Abbildung 61: Möglicher Lösungsweg im Hauptskript

Die Abbildung 62 zeigt dagegen das vollständige Programm. Es handelt sich dabei lediglich, wie zuvor, um ein Lösungsbeispiel.

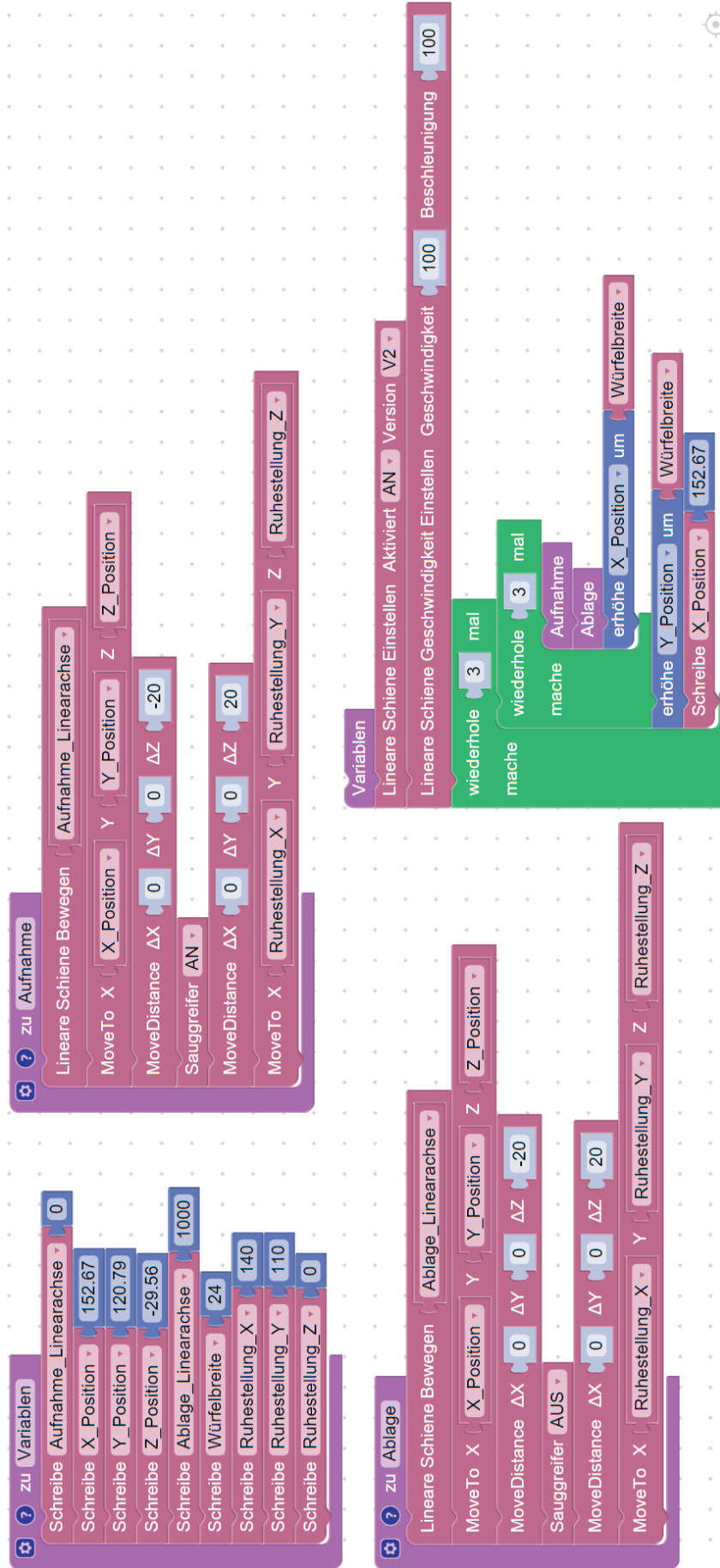


Abbildung 62: Musterlösung zum Übungsbeispiel

## 3.7 Nutzung der IO-Ports zur Vernetzung

### 3.7.1 Vernetzung mittels Taster

Jan Landherr, Dani Hamade, Carl von Ossietzky Universität Oldenburg

In der Industrie werden Roboter aus unterschiedlichen Gründen von Hand gesteuert. Auf der einen Seite kann zum Beispiel die Sicherheit von dem Personal besser gewährleistet werden, wenn der Roboter sich erst nach einem Tastendruck durch einen Anwender in Bewegung setzen kann. Auf der anderen Seite können so kompliziertere Prozesse besser zeitlich abgepasst werden, d. h. nachdem ein Roboter seine Tätigkeit beendet hat und nun für einen weiteren Arbeitsschritt einem anderen Roboter Raum geben muss, kann ein Taster an einem anderen Ort platziert werden, damit der vorige Roboter dem nächsten seinen Startzeitpunkt signalisiert. Auf diese Weise können Schäden durch Kollisionen vermieden werden. Kommunikation zwischen Robotern wird in den kommenden Abschnitten im Detail erklärt. Dieses Unterkapitel dient lediglich einem kurzen Ausblick auf den Anschluss eines Tasters an den Dobot Magician. In der NBC ist eine Bauanleitung für eine Tasterlösung enthalten, mit der zwei Taster in Verbindung mit zwei Dobot genutzt werden können.

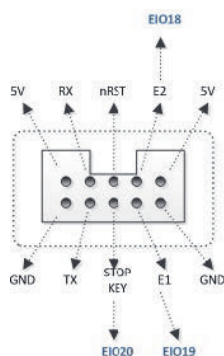
### 3.7.2 Vernetzung mittels EIO-Schnittstelle

Steffen Dreier, Evangelisches Gymnasium Nordhorn

Signale werden in der Digitaltechnik mit 0 ("LOW") oder 1 ("HIGH") kodiert. Diese Kodierung können wir z. B. mit einem Spannungslevel von 0 V oder 3,3 V realisieren. Dazu müssen die Schnittstellen entsprechend die Level-Input (Spannung wird gelesen) bzw. Level-Output (Spannung wird gesendet) Funktionalität beherrschen.

**Achtung: Viele Mikrokontroller sind sehr empfindlich hinsichtlich der erlaubten Spannung. Eine zu hohe Spannung zerstört den Mikrokontroller irreversibel.**

Für den den DOBOT Magician werden in der Dokumentation (Dobot Magician V2 User Guide V2.0.pdf) im Kapitel 4.3 Multiplexed I/O Interface Description die Eingabe/Ausgabe-Schnittstellen beschrieben. Die meisten Pins haben mehrfache Funktionsbelegungen. Im folgend vorgestellten Beispiel benutzen wir die Pins EIO18 und EIO19 sowie einen GND-Pin als Rückleiter aus dem "Communication Interface" auf der Rückseite des Dobot Magician. In der "I/O addressing"-Tabelle kann man erkennen, dass der PIN EIO18 den Level Output Modus jedoch nicht Level Input beherrscht. Der PIN EIO19 hingegen beherrscht nur den Level Input und nicht den Level Output. Ganz wichtig ist, dass beide PINs die gleiche "Voltage" haben! In diesem Fall also 3,3 V.



I/O Port	Spannung	Ausgang	PWM	Eingang	ADC
18	3,3 V	X	-	-	-
19	3,3 V	-	-	X	-
20	3,3 V	-	-	X	-

Ansicht und Beschreibung zum "Communication Interface" auf der Rückseite des Dobot Magician



### 3.7.2.1 Test mit einem DOBOT Magician

Mit Hilfe von einfachen female-DUPONT-Kabel werden die PINs EIO18 und EIO19 verbunden. Ein Kabel zum Verbinden der GND-PINs ist bei nur einem einzigen DOBOT Magician nicht notwendig, da alle GND PINs intern verbunden sind.

**Achtung:** Um Beschädigungen zu vermeiden, empfiehlt es sich, den DOBOT Magician auszuschalten und erst dann die PINs mit Kabeln zu verbinden.

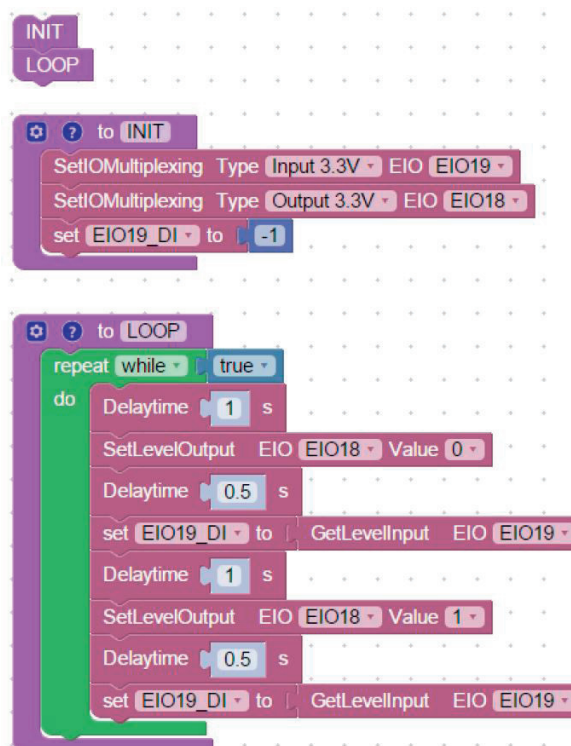
Zum Testen der Output-Input-Verbindung ist in Abbildung 3 ein DobotBlock Programm abgebildet. Dabei wird zunächst in der Funktion "INIT" eine Variable (hier: EIO19\_DI) mit dem Wert "-1" initialisiert. Später sollen die Werte 0 und 1 zur Kontrolle auf der Scratch-Bühne der Dobot-Block Programmieroberfläche visualisiert werden. Weiter werden mit dem Block **Set digital Output Port** der Port EIO19 für den Modus IOFunctionDI und der Port EIO18 für den Modus IOFunctionDO initialisiert.

**Achtung:** Wird der Modus nicht entsprechend der Hardwareleistungsfähigkeit der "I/O addressing Tabelle" initiiert, können Schäden am Roboter entstehen!

Im LOOP wird der Output-Port "EIO18" auf "LOW" **Set digital Output Port EIO18 Value LOW** und dann "HIGH" **Set digital Output Port EIO18 Value HIGH** gesetzt und jedes Mal zur Kontrolle entsprechend der Input-Port "EIO19" in die Variable "EIO19\_DI" ausgelesen **set EIO19\_DI to Get Digital Input Reading EIO19**. Die "wait" Blöcke **wait 1 seconds** sind entweder dafür da den Level-Wechsel nach dem "Set digital Output Port..." **Set digital Output Port EIO18** gesichert abzuwarten oder für den Anwender sichtbar zu machen (nach der Zuweisung an die Variable "EIO19\_DI"). Als Ergebnis sollte die Variable "EIO19\_DI" im Abstand von 1,5 sec abwechselnd den Inhalt "0" und "1" anzeigen.



Umsetzung in DobotBlock für einen DOBOT Magician

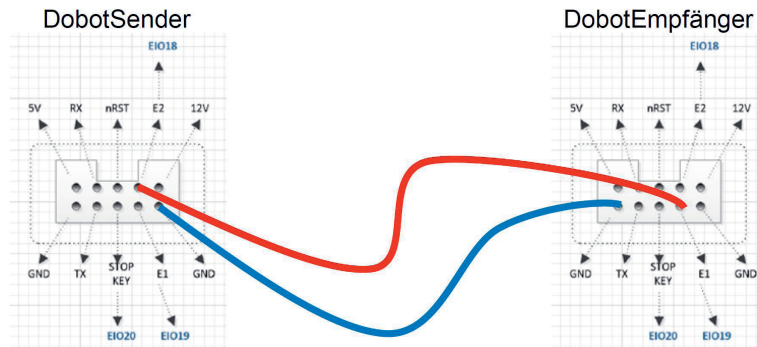


Umsetzung in DobotStudio Blockly

### 3.7.2.2 Test mit zwei DOBOT Magician

Die beiden DOBOT Magician Roboter "DobotSender" und "DobotEmpfänger" werden so verkabelt, dass der PIN "EIO18" des "DobotSender" mit dem PIN "EIO19" des "DobotEmpfänger" verbunden wird. Zusätzlich müssen ein GND PIN des "DobotSender" mit einem GND PIN des "DobotEmpfänger" mit einem Kabel verbunden werden. Im "Communication Interface" stehen dafür gleich zwei PINs zur Auswahl.

**Achtung:** Um Beschädigungen zu vermeiden, empfiehlt es sich, den DOBOT Magician auszuschalten und erst dann die PINs mit Kabeln zu verbinden.

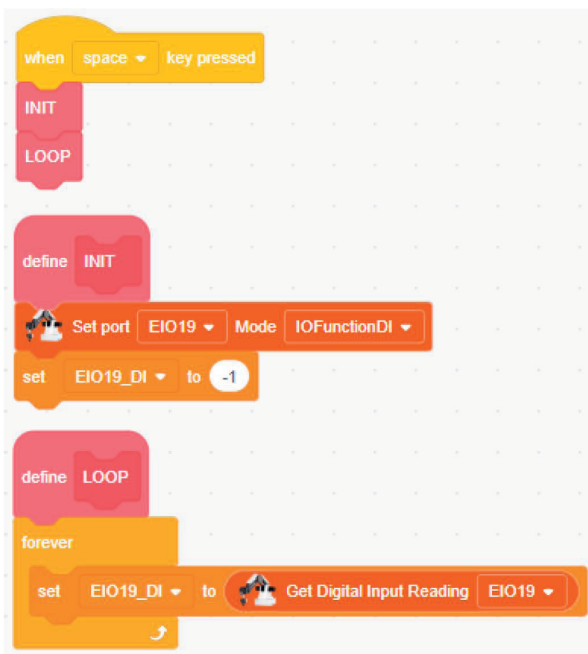


Verbindungskabel zwischen den beiden DOBOT Magician an die PINs des "Communication Interface"

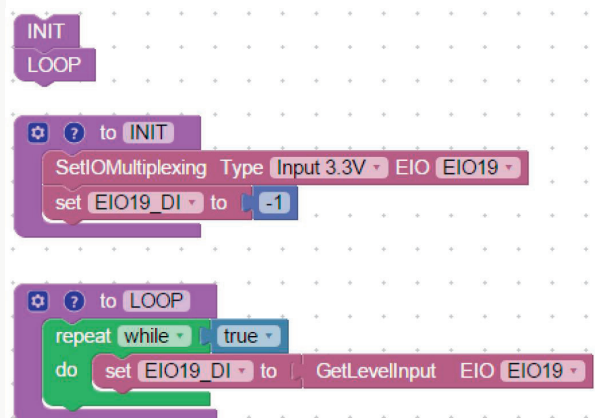
Hinweis: Natürlich ist es möglich mit einem zusätzlichen Kabel den "DobotEmpfänger" Port EIO18 mit dem "DobotSender" Port EIO19 zu verbinden, um auch vom "DobotEmpfänger" Signale in der Gegenrichtung an den "DobotSender" zu schicken.

Zum Testen der Output-Input-Verbindung zwischen den beiden DOBOTs wird das DobotBlock Programm aus dem Beispiel mit nur einem DOBOT Roboter auf zwei Programme aufgeteilt.

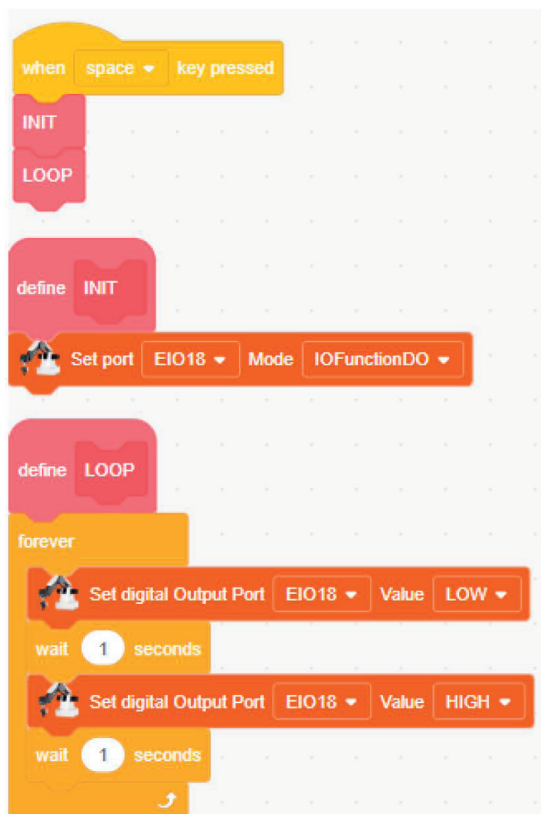
**Achtung!** Wird der Modus nicht entsprechend der Hardwareleistungsfähigkeit der "I/O addressing Tabelle" initiiert, können Schäden am Roboter entstehen!



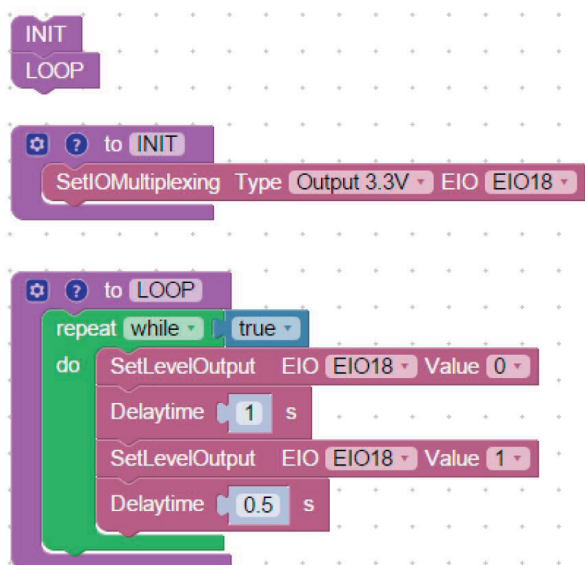
Umsetzung in DobotBlock für den "DobotEmpfänger"



Umsetzung in DobotStudio Blockly für den "DobotEmpfänger"



Umsetzung in DobotBlock für den "DobotSender"



Umsetzung in DobotStudio Blockly für den "DobotSender"

Nachdem zuerst das Programm des Empfängers und anschließend das Programm des Senders gestartet wurde, wird im Abstand von einer Sekunde abwechselnd der Inhalt "0" und "1" in der Variable "EIO19\_DI" auf der Scratch Bühne angezeigt.

### 3.7.3 Einführung in Python

Dani Hamade, Carl von Ossietzky Universität Oldenburg

Python ist eine höhere Programmiersprache, die sich in den vergangenen Jahren einer wachsenden Beliebtheit erfreut. Das hat zur Folge, dass ebenfalls die Anzahl an Anwendungsgebieten stetig zunimmt, sodass sie sich für beinahe jeden Zweck einsetzen lässt, sei es Webdesign, Datenbanken, Maschinelles Lernen uvm. Zur persönlichen Anwendung vieler Applikationen sind pythoninterne Bibliotheken verfügbar, die für den Gebrauch lediglich in die Umgebung importiert werden müssen, von wo sie aus frei einsetzbar sind. So ist der Dobot ebenfalls via Python steuerbar. Grundsätzlich gibt es eine firmeneigene Dobot-Python-API (*Application Programming Interface*), die sich allerdings nicht als sonderlich einsteigerfreundlich erweist, sodass eine eigene API entwickelt wurde. Diese basiert allein auf Python und muss zur Verwendung lediglich heruntergeladen und in der individuellen Python-Umgebung installiert werden (siehe Installationsinstruktionen). Die hier vorgestellten Befehle sind mithilfe dieser API ausführbar und werden im nächsten Kapitel erläutert. Im Gegensatz zu anderen Programmiersprachen arbeitet Python statt mit geschweiften Klammern mit Einrückungen, welches unter anderem die Lesbarkeit zur Folge hat, für die Python so geschätzt wird. Jede Missachtung von Einrückungen wird von Python mit konsequenter Fehlerausgabe bestraft, sodass bei der Programmierung besonders darauf geachtet werden muss.

#### 3.7.3.1 Import der Python-API und erste Anwendung

Prinzipiell müssen benötigte Bibliotheken importiert werden, welches der Übersichtlichkeit wegen zumeist am Anfang eines Python-Programms geschieht. Für den ersten Gebrauch des Dobots sind nur die Bibliotheken „sys“ und die API „pyDobot“ von Belang:

```
import sys
import pyDobot
```

Die sys-Bibliothek dient dazu, den aktuellen Pfad in der Dateistruktur um einen weiteren Ordner „src“ zu erweitern. Der Grund dafür ist, dass die API dort gelagert ist, auf welche natürlich der Zugriff benötigt wird. Mit folgendem Befehl wird dies realisiert:

```
sys.path.insert(0, "src")
```

Nun kann die eigentliche Verbindung von Python zum Dobot aufgebaut werden:

```
dBot = Dobot("COM3")
```

Mittels dieses Befehles wird -für den Python-Kenner- ein Objekt namens „Dobot“ erzeugt. Jeder Anwender, dem dieser Ausdruck nichts sagt, muss sich damit nicht weiter beschäftigen, solange der Eingabeparameter hinter dem „Dobot“-Aufruf mit der aktuellen Verbindung übereinstimmt. Bei Windows-Benutzern wird dies unter dem „Geräte-Manager“ > „Anschlüsse (COM & LPT)“ > „Silicon Labs CP210x [...]“ angezeigt. Der wichtige Teil davon, für den in die Anführungszeichen gesetzten Eingabeparameter, ist in der Klammer „COM[...]“ mit der entsprechenden Zahl zu finden. Bei UNIX-basierten Systemen, wie Mac und Linux, sind die Verbindungen unter „/dev/[...]“ gelistet, wobei die USB-Verbindung an sich als Name auftritt, beispielsweise:

```
dBot = Dobot("/dev/ttyUSB0")
```

Um die Verbindung zu lösen, wird (Betriebssystem-unabhängig) folgender Befehl verwendet:

```
dBot.close()
```

### 3.7.3.2 Pick-and-Place in Python

Als erstes Beispiel könnte ein Programm erstellt werden, um den Dobot ein Objekt aufheben und an einem anderen Ort wieder ablegen zu lassen. Um das angeschlossene Werkzeug benutzen zu können, muss es im Programm zunächst initialisiert werden:

```
dBot.setEndEffector(Dobot.ENDEFFECTOR_SUCTIONCUP)
```

Mit diesem Aufruf wird der Sauggreifer als angeschlossenes Werkzeug erklärt, wodurch der Dobot die Koordinaten weiß, wann mit dem Ende des Sauggreifer die Untersatz-Oberfläche erreicht ist.

Der Sauggreifer selbst lässt sich mit dem folgenden Aufruf steuern:

```
dBot.suctionCupOn(True)
```

Der Eingabeparameter True ist eine boolean-Variable und sorgt für die Einschaltung des Sauggreifers, während False den Sauggreifer ausschaltet.

Mithilfe von einigen Variablendeklarationen und dem jump-Befehl lässt sich der Dobot in einer JUMP-Bewegung zum gewünschten Ort transportieren:

```
dBot.jumpTo(X_Aufnahme, Y_Aufnahme, Z_Aufnahme, R_Aufnahme)
```

Das vollständige Programm könnte dann beispielsweise so aussehen:

```
import sys
import pyDobot

sys.path.insert(0, "src")

def Pick_Place():
    dBot = Dobot("COM3")
    dBot.setEndEffector(Dobot.ENDEFFECTOR_SUCTIONCUP)

    X_Aufnahme = 211.57
    Y_Aufnahme = -157.48
    Z_Aufnahme = -43.92
    R_Aufnahme = -36.66

    X_Ablage = 230.12
    Y_Ablage = 62.61
    Z_Ablage = -44.53
    R_Ablage = 15.22

    dBot.jumpTo(X_Aufnahme, Y_Aufnahme, Z_Aufnahme, R_Aufnahme)
    dBot.suctionCupOn(True)
    dBot.jumpTo(X_Ablage, Y_Ablage, Z_Ablage, R_Ablage)
    dBot.suctionCupOn(False)
    dBot.moveRel(0.0, 0.0, 10.00)
    dBot.close()
```

In diesem Beispiel wurde das Programm in eine Funktion eingebettet, kenntlich gemacht durch das def. Ähnlich wie bei Blockly werden die Funktionen nicht durch ihre bloße Existenz aufgerufen. Der Quellcode-Absatz zeigt lediglich nur eine Funktionsdeklaration. Der Aufruf von der Funktion könnte durch den uneingerückten Funktionsnamen samt Parameter stattfinden:

```
Pick_Place()
```

Wobei die elegantere Methode darin besteht eine if-Kondition aufzustellen, welche fragt, ob das Programm selbst aufgerufen wurde oder durch ein anderes Programm. Bei manchen Programmen möchte man möglicherweise nur einige Funktionen eines Skripts übernehmen. Bei einem Import würde das Programm aber auch jede Zeile, die nicht Teil einer Funktion ist, mit ausführen.

Um dies zu verhindern, wird folgendes hinzugefügt:

```
if __name__ == '__main__':
    Pick_Place()
```

Es gibt bei den Bewegungsbefehlen neben der oben gezeigten noch die Möglichkeit das über eine API-interne Variable zu realisieren, nämlich über *DobotPose*([*x, y, z, r*]). Dadurch kann das Programm ein wenig kürzer und damit ebenfalls übersichtlicher gestaltet werden. Um die Parameter allein und nicht das ganze Objekt zu übergeben, muss bei dem *jump*-Aufruf ein Asterisk (\*) vorgesetzt werden:

```
pose = DobotPose([230.12, 62.61, -44.53, 15.22])
dBot.jumpTo(*pose)
```

Der Einfachheit halber wurde im Folgenden mit den Standardvariablen statt mit dem *DobotPose*([]) -Objekt gearbeitet.

### 3.7.3.3 Schleifen in Python

Jeden Arbeitsschritt in einem Programm einzeln zu schreiben, ist nicht nur müßig, sondern verfehlt auch den Sinn des Programmierens vollständig, da es die Arbeit erleichtern soll. Wenn also regelmäßige Anpassungen vorgenommen werden müssen, lässt es sich mittels einer Schleife zusammenfassen, solange die Bedingungen klar formulierbar sind. Es gibt generell (in jeder Programmiersprache) *while*- und *for*-Schleifen. *While* bezeichnet den Schleifenprozess, welcher solange durchgeführt ist, bis ein Schlusskriterium (dem *while*-Aufruf anhängend) erreicht wurde:

```
zaehler = 0

while kriterium < zaehler:
    [...]
    zaehler = zaehler + 1
```

Die letzte Zeile lässt sich sogar verkürzen:

```
zaehler += 1
```

Auf diese Weise wird der Zähler automatisch um einen Wert erhöht. Dies lässt sich ebenso auf Multiplikation, Subtraktion und Division anwenden.

Die andere Schleife ist die *for*-Schleife, in welchem die Zählervariable automatisch angepasst wird. In Python wird das Start- und Abbruchkriterium bei einer simplen Zählung mit *range*([*Start,]Ende*) realisiert:

```
for zaehler in range(0,3):
    [...]
```

In einem Beispiel, in welchem der Dobot einen Turm aus drei nebeneinander liegenden Schaumstoff-Würfeln bauen soll, könnte das zum Beispiel so aussehen:

```
import time

def Turmbau():
    dBot = Dobot("COM3")
    dBot.setEndEffector(Dobot.ENDEFFECTOR_SUCTIONCUP)

    X_Aufnahme = 211.57
    Y_Aufnahme = -157.48
    Z_Aufnahme = -43.92
    R_Aufnahme = -36.66
    Wuerfelbreite = 24.5

    X_Ablage = 230.12
```

```

Y_Ablage = 62.61
Z_Ablage = -44.53
R_Ablage = 15.22

for i in range(3):
    dBot.jumpTo(X_Aufnahme, Y_Aufnahme, Z_Aufnahme, R_Aufnahme)
    dBot.suctionCupOn(True)
    dBot.jumpTo(X_Ablage, Y_Ablage, Z_Ablage, R_Ablage)
    dBot.suctionCupOn(False)
    dBot.moveRel(0.0,0.0,10.0)
    time.sleep(0.5)
    Y_Aufnahme += Wuerfelbreite
    Z_Ablage += Wuerfelbreite
dBot.close()

if __name__ == '__main__':
    Turmbau()

```

Dieses Programm wurde wieder in Form einer Funktion zur besseren Übersichtlichkeit definiert, welches in der sogenannten „main“-Funktion aufgerufen wird. Hier wurde nun für eine Verzögerung, ähnlich zu dem „Delaytime“ in Blockly eingebaut. Da Python keine eigene Funktion dafür besitzt, wird die Bibliothek *time* importiert und aufgerufen (*time.sleep([...])*).

Um das Dobot-Objekt nicht wiederholt aufrufen zu müssen, kann das Objekt bei Funktionsaufruf als Parameter übergeben werden. Dafür muss es in der Funktion als Eingabeparameter in die Klammer nach dem Funktionsnamen definiert und bei Aufruf (durch die main) eingesetzt werden:

```

def Turmbau(Dobot: dobot):
    dobot.setEndEffector(Dobot.ENDEFFECTOR_SUCTIONCUP)

    X_Aufnahme = 211.57
    Y_Aufnahme = -157.48
    Z_Aufnahme = -43.92
    R_Aufnahme = -36.66
    Wuerfelbreite = 24.5

    X_Ablage = 230.12
    Y_Ablage = 62.61
    Z_Ablage = -44.53
    R_Ablage = 15.22

    for i in range(3):
        dobot.jumpTo(X_Aufnahme, Y_Aufnahme, Z_Aufnahme, R_Aufnahme)
        dobot.suctionCupOn(True)
        dobot.jumpTo(X_Ablage, Y_Ablage, Z_Ablage, R_Ablage)
        dobot.suctionCupOn(False)
        dobot.moveRel(0.0,0.0,10.0)
        time.sleep(0.5)
        Y_Aufnahme += Wuerfelbreite
        Z_Ablage += Wuerfelbreite

if __name__ == '__main__':
    dBot = Dobot("COM3")
    Turmbau(dBot)
    dBot.close()

```

Auf diese Weise kann das *Dobot*-Objekt für jede Funktion verwendet werden, ohne einzeln erzeugt werden zu müssen. Dass das Objekt in der *Turmbau()*-Funktion anders heißt, ist möglich, da es bei Funktionsdeklaration in der Klammer entsprechend definiert wurde. Es muss nicht zwangsläufig *dBot* heißen. Bei der Funktion fällt aber noch das *Dobot*: auf. Dies ist optional und dient dazu, in den IDEs die automatische Vervollständigung zu ermöglichen, welche andernfalls nicht funktionieren würde. Der Funktion wird damit vermittelt, dass es sich bei der genannten Variable um das *Dobot*-Objekt handelt, worauf sich wiederum die automatische Vervollständigung bezieht.

### 3.7.3.4 Nutzung des Förderbandes und des RGB-Farbsensors

Mithilfe der Python-Dobot-API lässt sich auch das angeschlossene Förderband steuern. Es muss nach Erzeugung des Dobot-Objektes initialisiert werden:

```
dBot.setConveyorPort(DobotPort.STEPPER1)
```

Die Einstellung der Geschwindigkeit (in mm/s) ist ähnlich wie in Blockly:

```
dBot.setConveyorSpeed(50)
```

Nun muss eine Verzögerung und das Setzen der Geschwindigkeit auf 0 folgen. Insgesamt könnte dies so aussehen:

```
dBot.setConveyorSpeed(50)
time.sleep(6.0)
dBot.setConveyorSpeed(0)
```

Einen Dobot nacheinander drei Würfel aufnehmen und auf das Förderband setzen zu lassen, welches sich daraufhin in Bewegung setzt, könnte, wie folgt, aussehen:

```
def Foerderband():
    dBot = Dobot("COM4")
    dBot.setEndEffector(Dobot.ENDEFFECTOR_SUCTIONCUP)
    dBot.setConveyorPort(DobotPort.STEPPER1)

    X_Aufnahme = 224.28
    Y_Aufnahme = -31.18
    Z_Aufnahme = -50.94
    X_Ablage = -81.39
    Y_Ablage = -213.91
    Z_Ablage = 8.59

    numLoops = 3
    Wuerfelbreite = 24.5

    for k in range(0, numLoops):
        dBot.jumpTo(X_Aufnahme, Y_Aufnahme, Z_Aufnahme)
        dBot.suctionCupOn(True)
        dBot.jumpTo(X_Ablage, Y_Ablage, Z_Ablage)
        dBot.suctionCupOn(False)

        dBot.setConveyorSpeed(50)
        time.sleep(6.0)
        dBot.setConveyorSpeed(0)
        Y_Aufnahme += Wuerfelbreite

    dBot.close()
```

Ähnlich ist es beim Farbsensor, welcher zunächst initialisiert werden muss:

```
dBot.setColorSensor(True, DobotPort.GP2)
```

Die Farbabfrage geschieht ähnlich wie bei Blockly:

```
dBot.getColor()
```

Die einzelnen Farben werden mit einem Punkt abgetrennt in einer if-Anweisung abgefragt:

```
if dBot.getColor().g:    ODER    if dBot.getColor().g == True:
```



Die if-Anweisung ist der in Blockly ähnlich. Auf das if folgt die zu prüfende Variable und die Prüfbedingung wird danach angehängt. Prüfbedingungen können „<“ (kleiner als), „>“ (größer als), „<=“ (kleiner oder gleich wie), „>=“ (größer oder gleich wie), „!=“ (ungleich) oder eben auch „==“ (gleich) sein. Um jeweils auf eine bestimmte Bedingung zu prüfen, kann auf ein „if“ ein „elif“ folgen: Wenn es nicht die „if“-Bedingung erfüllt, dann evt. die „elif“-Bedingung. Im Fall, dass keine der Bedingungen zutrifft, kann die Befehlskette nach „else“ die entsprechenden Aktionen einleiten.

Es gibt ebenfalls die Möglichkeit auf mehrere Bedingungen zu prüfen. Falls rot oder grün erkannt werden, führe etwas aus:

```
if dBot.getColor().g or dBot.getColor().r:
```

Dies wird entsprechend mit einem or zwischen zwei Konditionen realisiert. Falls zwei Bedingungen zutreffen müssen, müsste das or durch ein and ersetzt werden. Bei Farbsortierung sind mehrere Bedingungen vielleicht nicht so sinnvoll, allerdings ist es vorteilhaft diese Methode kennengelernt zu haben. Wenn nun ein Dobot, wie bei einer Produktionskette, durch das Förderband transportierte Würfel, aufnehmen, analysieren und den Farben entsprechend sortieren soll, könnte das folgendermaßen aussehen.

```
def Aufgabe4_2_3():
    dBot = Dobot("COM4")
    logging.basicConfig(level=logging.INFO)
    dBot.setEndEffector(Dobot.ENDEFFECTOR_SUCTIONCUP)
    dBot.clearAlarms()

    X_Aufnahme = -29.50
    Y_Aufnahme = -195.02
    Z_Aufnahme = 9.72
    R_Aufnahme = -98.60

    X_RGB = 44.82
    Y_RGB = -283.02
    Z_RGB = 35.92
    R_RGB = 15.22

    X_Rot = 182.12
    Y_Rot = -169.60
    Z_Rot = -41.43

    X_Blau = 225.59
    Y_Blau = 37.97
    Z_Blau = -50.35

    X_Gruen = 209.97
    Y_Gruen = -64.49
    Z_Gruen = -50.77

    X_Farblos = -29.50
    Y_Farblos = -195.02
    Z_Farblos = 9.72

    for i in range(3):
        time.sleep(5.0)
        dBot.jumpTo(X_Aufnahme, Y_Aufnahme, Z_Aufnahme, R_Aufnahme)
        dBot.suctionCupOn(True)
        dBot.jumpTo(X_RGB, Y_RGB, Z_RGB, R_RGB)

        time.sleep(2.0)

        dBot.setColorSensor(True, DobotPort.GP2)
        if dBot.getColor().g:
            print("Gruen")
            dBot.jumpTo(X_Gruen, Y_Gruen, Z_Gruen)
```

```

elif dBot.getColor().r:
    print("Rot")
    dBot.jumpTo(X_Rot, Y_Rot, Z_Rot)
elif dBot.getColor().b:
    print("Blau")
    dBot.jumpTo(X_Blau, Y_Blau, Z_Blau)
else:
    print("Weder rot noch blau noch gruen")
    dBot.jumpTo(X_Farblos, Y_Farblos, Z_Farblos)

dBot.suctionCupOn(False)
dBot.moveRel(0.0,0.0,10.0)

dBot.close()

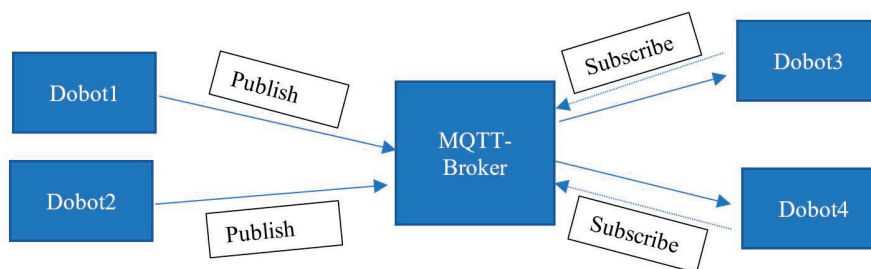
```

### 3.7.3.5 Vernetzte Produktion mit Python (MQTT)

Damit man nun mehrere DobotS miteinander vernetzen kann, ohne sie direkt miteinander verdrahten zu müssen wie bei dem Optokoppler oder dem Taster, kann man standardisierte Nachrichtenprotokolle wie MQTT zur Machine to Machine Kommunikation (M2M) nutzen. Diese Funktion wurde in der Python API für den Dobot implementiert. MQTT ist hierbei ein Standard-Messaging-Protokoll für das Internet of Things (IoT).

*„Das Messaging-Protokoll MQTT (Message Queueing Telemetry Transport) kommt vor allem im Bereich des Internet Of Things (IOT) zum Einsatz. Ziel bei der Entwicklung dieses Protokolls war das Funktionieren in unzuverlässigen Netzwerken mit geringer Bandbreite mit Geräten, die nur über eingeschränkte Ressourcen (wie geringem Speicher) verfügen und verhältnismäßig kurze Nachrichten übertragen.“ (Abts, 2022, S. 187).*

Grundlegend beinhaltet die MQTT-Struktur sogenannte Publisher, einen Broker und Subscriber (siehe Abbildung 68). Die Publisher sind in unserem Anwendungsfall DobotS, die Nachrichten versenden. Einer oder mehrere andere DobotS können diese Nachrichten als Subscriber, welche durch den MQTT Broker von den Publishern entkoppelt sind, empfangen. Die Nachrichten, die vom Publisher versendet werden gehen an ein bestimmtes Topic (z.B. Farberkennung). Je nach dem, für welches Topic die Subscriber angemeldet sind, werden sie über den Broker an diese weitergeleitet.



Als Broker verwenden wir in diesem Fall den Open Source MQTT-Broker „Eclipse Mosquitto“. Eine detaillierte Installationsanleitung erhält man hier: <https://mosquitto.org/download>.

Jeder Dobot der nun eine Verbindung zum Broker aufbaut, ist ein Client mit einer bestimmten ID. Diese ClientID sollte sorgfältig gewählt werden, sodass die DobotS hinterher gut voneinander zu unterscheiden sind. Im Python Code wird dies wie folgt festgelegt (hier „myDobot“):

```

# intern
import sys
import time
# own
sys.path.insert(0, "src") # Diese Zeile ist nur notwendig, wenn
pyDobot nicht installiert wurde.
from pyDobot import DobotMqtt, DobotPort, DobotIOFunction

```

```

if __name__ == "__main__":
    dBot = DobotMqtt(
        mqtt_host      = "localhost",
        mqtt_auth      = ("mqttuser", "mqttpasswd"),
        mqtt_client_id = "myDobot",
        serial_port     = "COM3"
    )

```

Zunächst werden oben die notwendigen Bibliotheken importiert. Im Hauptprogramm beginnen dann direkt schon die Definitionen für die MQTT-Struktur. Wir arbeiten in diesem Beispiel zunächst über einen lokalen Host (also der Laptop als Broker). Zur Autorisierung kann man bei Mosquitto nun ein Passwort und einen Username festlegen (hier „mqttuser“ und „mqttpasswd“). Die Client ID ist diejenige, welche auch im Broker angezeigt wird (hier myDobot). Zum Schluss muss man nur noch den Port definieren (der große Vorteil ist nun, dass man beliebig viele Dobots (z. B. über einen USB HUB) an einen Laptop anschließen kann). Als Erweiterung kann man hier auch einen Raspberry PI als Broker schalten, sodass der Laptop nicht zwangsweise permanent bei den Doboten stehen muss. Man kann die Doboten dann über WLAN und den Raspberry steuern. Anhand eines Beispiels soll nun gezeigt werden, wie der Nachrichtenaustausch abläuft. Die Problemstellung ist, dass zwei Doboten, die in einer Fertigungslinie sehr nah aneinander stehen, ein Homing bei Schichtbeginn ausführen sollen. Da die Doboten allerdings so nah beieinander stehen, müssen sie das Homing zur Kollisionsvermeidung nacheinander durchführen. Dobot1 soll in diesem Fall also eine Referenzfahrt durchführen und erst nach Abschluss eine Nachricht an den Broker übermitteln, dass er fertig ist, sodass der zweite Dobot (Subscriber) starten kann. Es ergibt sich folgender Code:

#### Für Dobot1:

```

import sys
import time
# own
sys.path.insert(0, "src") # Diese Zeile ist nur notwendig,
wenn pyDobot nicht installiert wurde.
from pyDobot import DobotMqtt, DobotPort, DobotIOFunction

if __name__ == "__main__":
    dBot = DobotMqtt(
        mqtt_host      = "localhost",
        mqtt_auth      = ("mqttuser", "mqttpasswd"),
        mqtt_client_id = "Dobot1",
        serial_port     = "COM3"
    )

    dBot.clearAlarms()
    dBot.enableRemoteCtrl(True)
    dBot.home()
    dBot.sendMqttMsg ("fertig")
    dBot.close()
    sys.exit()

```

#### Für Dobot2:

```

# intern
import sys
import time
# own
sys.path.insert(0, "src") # Diese Zeile ist nur notwendig,
wenn pyDobot nicht installiert wurde.
from pyDobot import DobotMqtt, DobotPort, DobotIOFunction

if __name__ == "__main__":
    dBot = DobotMqtt(
        mqtt_host      = "localhost",
        mqtt_auth      = ("mqttuser", "mqttpasswd"),

```

```
        mqtt_client_id = "Dobot2",
        serial_port    = "COM3"
    )

    dBot.clearAlarms()
    dBot.enableRemoteCtrl(True)
    msg=dBot.getMqttMsg()
    if msg ['text'] == "oki":
        dBot.home()

    dBot.close()
    sys.exit()
```

Nun könnte man beliebig viele Nachrichten zwischen den Doboten austauschen (z. B. könnte Dobot2 nach dem Homing den Arbeitsprozess initiieren, indem er wieder eine Nachricht sendet, die Dobot1 empfängt). Außerdem ist es möglich, nur auf Nachrichten eines bestimmten Absenders zu reagieren (durch Ergänzung eines „from“ in der Bedingung). Will man, dass die Doboten zu einem bestimmten „Topic“ (also Oberthema) eine Handlung ausführen, so kann man „publishen“ und „subscriben“. Ein Beispieltopic wäre hier die Farberkennung. Der Publisher müsste folgende Daten veröffentlichen:

```
dBot.mqttPubMsg("Farbe", dBot.getColor())
```

Der andere Dobot muss diese Daten entsprechend subscriben:

```
dBot.mqttSubMsg("Farbe", dBot.getColor())
```

## Literatur

Abts, D. (2022): *Masterkurs Client/Server-Programmierung mit Java. Anwendungen entwickeln mit Standard-Technologien. 6. Auflage.* Wiesbaden: Springer Fachmedien GmbH.

## 3.8 Vergleich der Programmieroptionen, Möglichkeiten in DobotLab

Marvin Bersiner, robospace gGmbH

In diesem Abschnitt wird die Software DobotLab kurz vorgestellt. Im Anschluss gibt es eine Auflistung der Fehler und die jeweiligen Work-Arounds der drei bestehenden Dobot Softwareanwendungen DobotStudio, DobotBlock und DobotLab. Abschließend werden noch die Vorteile bei der Nutzung von DobotLab als Hauptsoftwareanwendung aufgezeigt.

### 3.8.1 Allgemeine Vorstellung DobotLab

DobotLab ist eine integrierte Software Plattform, die webbasiert via <https://dobotlab.dobot.cc> erreicht oder desktopbasiert als Programm auf dem Laptop oder PC installiert werden kann. Das folgende Bild zeigt den Startbildschirm von DobotLab. Getestete Software, sowie weitere Schulungsunterlagen sind unter <https://robospace.de/dobot-schulung> erreichbar. Die aktuellste Software und ein User Guide sind über die offizielle Website von Dobot <https://dobot.cc> unter Support > Download Center verfügbar.

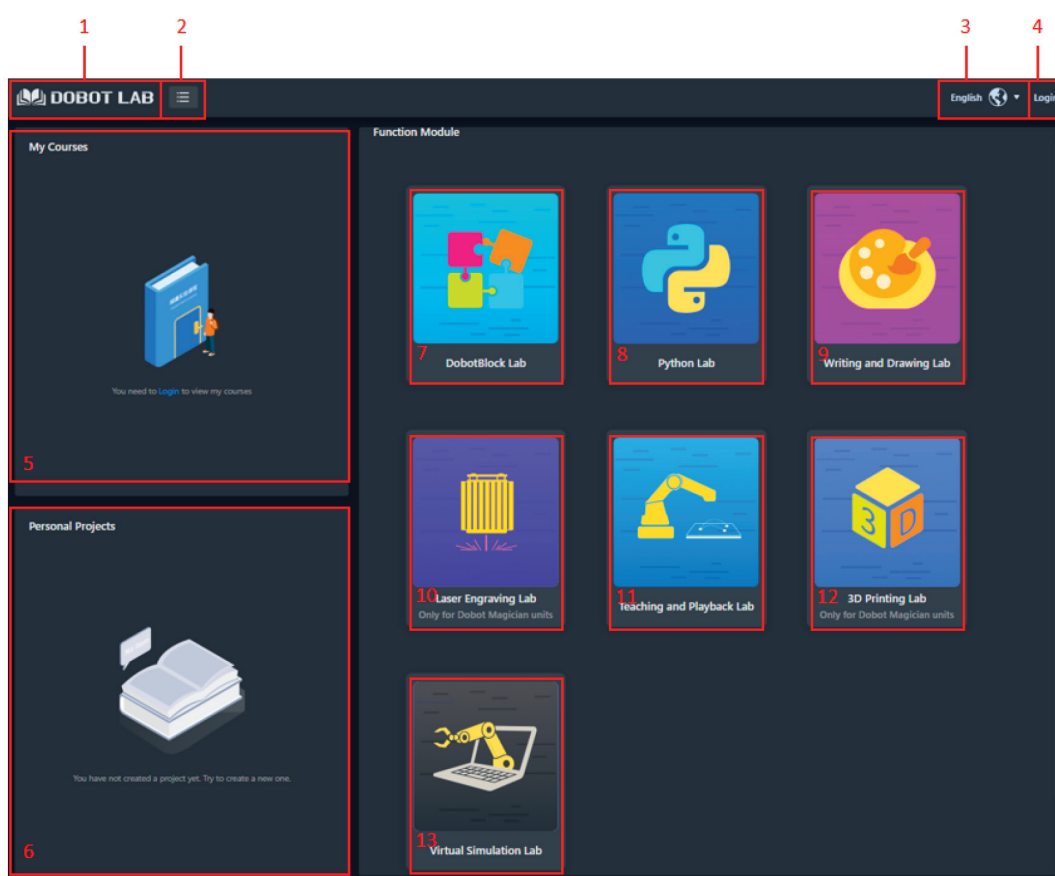


Abbildung 1: Übersicht DobotLab

Nr.	Modul	Beschreibung
1	Startseite	Klicken, um zur DobotLab Startseite zu gelangen
2	Menü	Currency: Anleitung zur Nutzung von DobotLab Help: Anzeigen und Download von Dokumenten für DobotLab Feedback: Feedback über DobotLab weitergeben About: Informationen über DobotLab
3	Sprache	Auswahl der Sprache
4	Login	Einloggen mit dem eigenen Account
5	Meine Kurse	Angebot von frei-zugänglichen Online-Kursen
6	Persönliche Arbeiten	Anzeigen und bearbeiten von gespeicherten Projekten
7	DobotBlock Lab	Bedienung des Dobot durch blockbasiertes Programmieren

Nr.	Modul	Beschreibung
8	Python Lab	Bedienung des Dobot durch skriptbasiertes Programmieren
9	Writing and Drawing Lab	Bewegung des Dobot um zu Schreiben oder zu Malen
10	Laser Engraving Lab	Steuerung des Dobot zum Gravieren eines Bitmap-Bildes mit einem Laser
11	Teaching and Playback Lab	Den Dobot anlernen, wie er sich bewegen soll. Aufzeichnung der Bewegung, Dobot führt die aufgezeichnete Bewegung aus.
12	3D Printing Lab	Bedienung des Dobot Magician um 3D zu drucken
13	Virtual Simulation Lab	Simulation von Roboterbewegungen durch Programmieren innerhalb einer virtuellen Umgebung und eines Roboter Modells

Im Folgenden wird nur auf Aspekte eingegangen, die sich zu den Softwareanwendungen DobotStudio oder DobotBlock unterscheiden.

### 3.8.2 Python Lab – Skriptbasierte Programmierumgebung

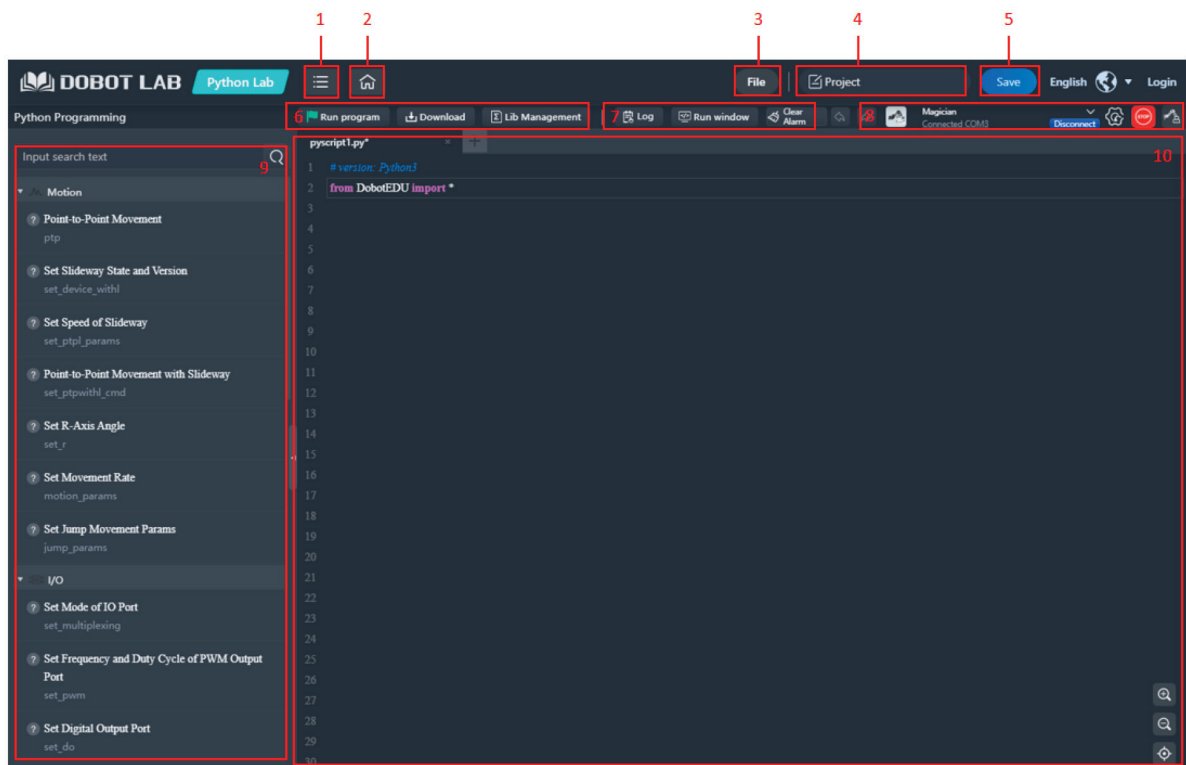


Abbildung 2: Startseite PythonLab

Nr.	Modul	Beschreibung
1	Menü	Currency: Anleitung zur Nutzung von DobotLab Help: Anzeigen und Download von Dokumenten für DobotLab Feedback: Feedback über DobotLab weitergeben About: Informationen über DobotLab
2	Home	Klicken, um zur DobotLab Startseite zu gelangen
3	Datei	Beinhaltet die Funktionen: Neue Datei, Öffnen, Speichern als, Hochladen von Lokal,..
4	Projekt Name	Anzeige des aktuellen Projektnamens
5	Speichern	Aktuelles Projekt speichern

Nr.	Modul	Beschreibung
6	Programm Kontrolle	„Run program“: Klicken um das aktuelle Programm im Code Bereich zu starten „Download“: Downloaden des aktuellen Programms auf den Speicher des Roboters „Lib Management“: Installieren von Python Erweiterungs-Bibliotheken. Danach können die Bibliotheks-Funktionen aufgerufen werden.
7	Lauf Informationen	Log: Alarm Infos werden angezeigt. Clear Alarm: Um die Fehlermeldungen zu löschen. Run window: Zeigt die aktuelle Laufzeit Aktivitäten
8	Geräte Kontrolle	Connect: Auswahl eines Gerätes und Aufbauen einer Verbindung mit dem Gerät Stop: Drücken, um im „Notfall“ das Programm zu stoppen Roboter-Symbol: Kontrolle/Bewegung des Dobot Magician
9	Kommando Liste	Anzeige der zur Verfügung stehenden Befehle. Doppelklick auf die Befehle, damit sie im Code Bereich erscheinen. Beim Klicken auf das Fragezeichen erscheint die Dokumentation zu den jeweiligen Befehlen.
10	Code Bereich	Bearbeitung von Programmen mithilfe von Python

**Anmerkung:** Bei der Verwendung von externen Geräten wie Sensoren oder dem Förderband sollte beim Verbinden mit dem Roboter „Magic Box + Magician“ ausgewählt werden. Erst dann werden in der linken Spalte alle notwendigen Befehle für die externen Geräte angezeigt.

### 3.8.3 Writing and Drawing

Die folgende Abbildung zeigt die Startseite des Writing and Drawing Programms. In dem rot markierten Feld kann der Text eingegeben werden, der von dem Dobot Magician gezeichnet werden soll. Der Text oder das Bild, welches gemalt werden soll, sollte sich in dem eingezeichneten Bereich befinden, ansonsten kann der Roboter die Zeichnung nicht abfahren, da er sich außerhalb seiner Achsgrenzen befindet.

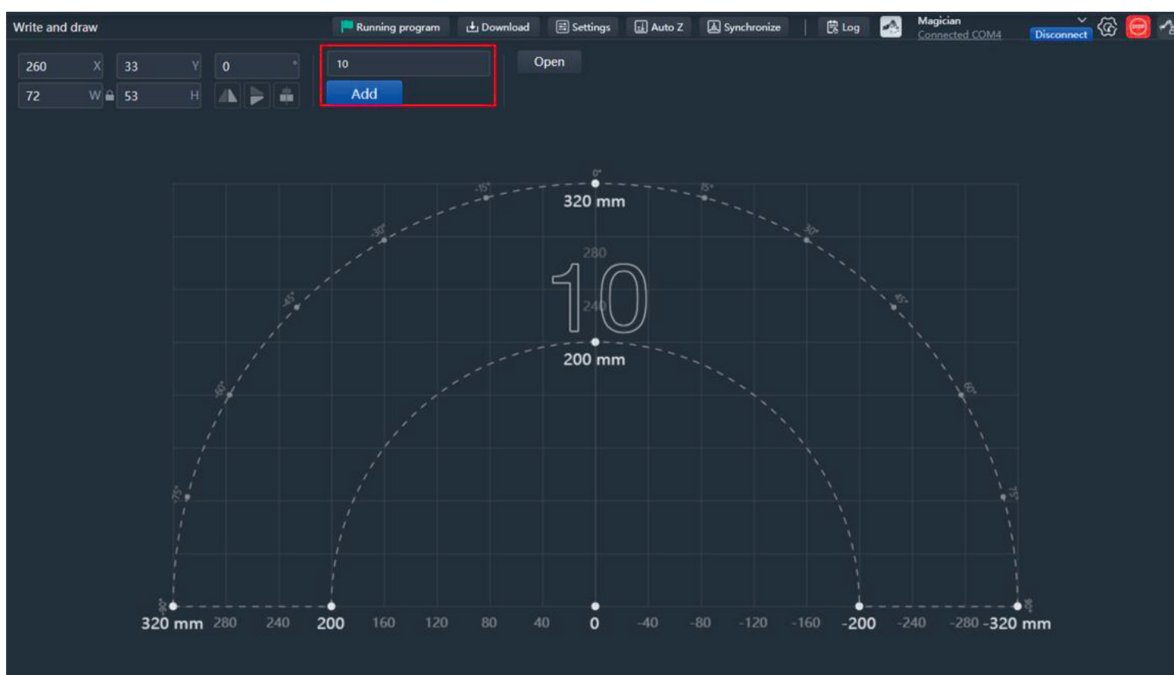


Abbildung 3: Startseite Write and Draw

Unter „Settings“ kann vor der Benutzung des Stiftes die Stiftanhebe-Höhe eingegeben werden. Diese liegt standardmäßig bei 20 mm. Bevor die Zeichnung abgefahren wird, muss der Stift einmal am Papier angesetzt werden. Wenn dies geschehen ist, kann der Button „Auto Z“ gedrückt werden, um die aktuelle Z-Position zu speichern.

Mithilfe der Linearachse können auch längere Texte oder Zeichnungen angefertigt werden. Hierfür muss zunächst der Roboter auf der Linearachse befestigt werden. Bei Klicken des Roboter-Symbol (rechts oben) öffnet sich das „Arm Control Panel“. In diesem muss für das Aktivieren der Linearachse bei „Rail“ das Häkchen gesetzt werden. Nun ist die Linearachse aktiv und der Arbeitsraum des Roboters hat sich verändert. Es können längere Grafiken dargestellt werden.

### 3.8.4 Fehlerliste der Hardware Dobot Magician und der Softwareanwendungen DobotStudio, DobotBlock und DobotLab

Fehler	Tritt auf	Work-Around
Hardware		
Ausschalter schaltet den Roboter nicht aus	Wenn der Roboter abstürzt	Stromkabel trennen
Stepper Motoren des Förderbands bewegen sich nicht korrekt	Wenn man zu hohe oder zu niedrigere Geschwindigkeiten einstellt	Mit einem Wert im Bereich von 50 mm/s starten und sich langsam an die Grenzen tasten
DobotStudio: Allgemein		
Fehlermeldung auf Chinesisch	Wenn der falsche Port beim Verbinden ausgewählt wird	Richtigen Port auswählen
Notaus funktioniert nicht	Wenn der Roboter abstürzt/aufhängt/nicht reagiert	Stromkabel trennen
DobotStudio: Teach and Playback		
Spalte für das Werkzeug / Linearachse wird nicht angezeigt	Wenn das Tool ausgewählt wird und vorher keins (oder der Stift) ausgewählt war	Neuen Punkt erstellen
Spalten/Positionen haben keinen Wert für die Werkzeugspalte	Wenn Positionen erstellt wurden und danach ein anderes Werkzeug ausgewählt wird	Doppelklick auf das entsprechende Feld und Zustand auswählen
DobotStudio: Blockly		
Alle Blöcke werden ausgeführt	Wenn man Blöcke vom Hauptstrang „abzieht“ und irgendwo in der Oberfläche positioniert	Blöcke einzeln deaktivieren
Man kann nicht mehrere Blöcke auf einmal deaktivieren		
Übersetzung von „Movements“ als „Antrag“		
Greifer dreht sich ungewollt	Wenn man den Springe zu/Jump to Block verwendet	
Es gibt keine Möglichkeit einer Joint Bewegung mit kartesischen Koordinaten	Wenn man Rechtsklick -> Add Point verwendet	
Förderband läuft nicht	Bei zu hohen Werten	Geschwindigkeitsrampe programmieren
„Photoelektrischer Sensor Erhalten“ gibt keinen booleschen Wert, sondern eine Zahl zurück		In einer Variable zwischenspeichern



Fehler	Tritt auf	Work-Around
Sensorausgangsblöcke sind nicht in jedem optisch passenden Block verwendbar		In einer Variable zwischenspeichern
Dataentypen können optisch nicht unterschieden werden		Ausprobieren, ob der Block an den Eingang andockt
Man kann die Linearachse nicht gleichzeitig mit dem Arm bewegen		
Man kann die Linearachse nicht mit einer vorgegebenen Geschwindigkeit steuern		
<b>DobotBlock</b>		
Linearachse kann nicht verwendet werden	Wenn der Geschwindigkeitsblock der Linearachse verwendet wird	Anderen Block verwenden oder DobotBlock v1.6.0-beta9 benutzen
Block zu Manipulation der Geschwindigkeit und Beschleunigung funktioniert nicht mehr	Bei Verwendung von DobotBlock v1.6.1-beta.4	DobotBlock v1.6.0-beta9 benutzen
Linearachse und Sensoren funktionieren nicht	Wenn mehrere Roboter in einem Programm verwendet werden	Alle Sensoren und Linearachsen an dem Bot anbringen, von welchem aus das Programm gestartet wird
<b>DobotLab: Allgemein</b>		
Kann Wifi-Modul verbinden, aber im Nachgang nicht mehr finden		Nutze DobotStudio, damit findest du den Roboter im Netz
In Python Lab: Kein Befehl <i>get_endeffektor</i>		
In Python Lab: Kein Befehl für die Arc-Bewegung		
DobotBlock Lab: Knopf zum Aufrufen des Python-Codes nicht immer verfügbar	Wenn man sich beim Öffnen von DobotLab mit dem Internet verbunden ist	DobotLab ohne Internet öffnen, dann erscheint der Python Button

### 3.8.5 Vorteile bei der Nutzung von DobotLab

Die folgende Auflistung zeigt eine Übersicht der Vorteile von DobotLab gegenüber der Nutzung von DobotBlock oder DobotStudio.

- Als Desktop und Web Anwendung nutzbar
- Teaching, Blockbasierte und Textbasierte Programmierung in einem Programm möglich
- Simulierte Umgebung vorhanden
- Viele Fehler aus den Programmen DobotStudio und DobotBlock wurden behoben
  - Bspw. können nun bei der Ansteuerung von zwei oder mehr Robotern die Sensoren und externen Komponenten an beliebigen Robotern angeschlossen werden.
- Bessere Dokumentation innerhalb der skriptbasierten Programmierumgebung