
# Xylobot 2.0

## Dein Persönlicher Xylophon Lehrer

Ein Lego Mindstorms Projekt von

Celikkol, Kang, Semercioglu, Vetrovska

FAU
Friedrich-Alexander-Universität
Erlangen-Nürnberg

# Xylobot 2.0

## Dein Persönlicher Xylophon Lehrer

von

Celikkol, Kang, Semercioglu, Vetrovska


| | |
|---|---|
| Mitglieder: | Dila Su Celikkol (23017887),<br>Si-Hoon Kang (22960060),<br>Batuhan Semercioglu (23047223),<br>Aneta Vetrovska (23041871), |
| Projekt Dauer: | April 2024 - Juli 2024 |
| Department: | Informatik |
| Lehrstuhl: | Hardware-Software-Co-Design |
| Betreuer: | Dr.-Ing. Stefan Wildermann |

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

# Inhaltsverzeichnis

# 1

# Einführung

Viele werden es aus ihrer Kindheit kennen, man findet zufällig ein Xylophon, das zu Hause herumliegt und versucht natürlich Lieder, die man kennt, darauf zu spielen. Aber sobald man anfängt zu spielen, stellt man mit Enttäuschung fest, dass es doch gar nicht so leicht ist wie gedacht und sich alle Noten komisch anhören. Oh wie schön es wäre, einen Lehrer beiseite zu haben, der mir das Xylophon vorspielen kann! Geanu diesen Kindheitswunsch in Erfüllung zu bringen, war unsere Idee, die wir für das Lego Mindstorms Praktikum unserer Universität FAU realisieren wollten. Das Praktikum bestand darin, von Grund auf einen eigenen, auf LEGO MINDSTORMS basierenden Roboter sowohl zu bauen als auch zu programmieren. Unser Ziel also: ein Roboter, der eigenständig Lieder auf dem Xylophon spielen kann, genannt Xylobot 2.0! Neben unserem Projekt gab es auch viele weitere interessante Ideen von anderen Gruppen: Rubik's-Cube-Solver, ein von Micromouse inspirierter Maze-Solver oder sogar ein Drucker, der Morse-Code ins Alphabet übersetzen und diesen drucken kann. Man könnte sagen, all diese Projekte haben die Gemeinsamkeit, dass ihre Umsetzung einer eindeutigen Aufgabe zugrunde liegt: „Löse den Rubik Würfel“ „Löse das Labyrinth“ oder „Übersetze den Morse-Code und drucke es als Alphabet-Buchstaben aus“. Musik jedoch, ist eine komplexe Zusammensetzung aus vielerlei Noten auf unterschiedlichen Stufen, den Pausen dazwischen, diversen Instrumenten, Spielern, Dirigenten und ihrer Fähigkeit für Zusammenarbeit. Demnach unterliegt die Aufgabe „Spiele Musik auf dem Xylophon“ vielseitigen Interpretationen, unter denen wir unsere eigene herausarbeiten mussten. Genau diese Interpretation, was es für uns *bedeutet*, einen mechanischen Roboter Musikstücke auf dem Instrument Xylophon spielen zu lassen, ist die interessante Herausforderung, die unser Projekt von den anderen unterscheidet. Dies umfasst v.a. die Verteilung einer großen Aufgabe auf mehrere unabhängige Komponenten sowie der dynamischen Koordinierung ihrer Zusammenarbeit, sowohl auf der mechanischen Hardware- als auch der logischen Software-Ebene. Diese Dokumentation berichtet über den Entwicklungsprozess des Projekts und erklärt die Funktionsweise des Roboters. Nun lasst uns beginnen, unser Xylobot zum Leben zu erwecken...

# 2

# Plan und Ablauf

## Projektplan

Zunächst soll der anfangs ausgearbeitete Projektplan beschrieben werden. In diesem wurde als Hauptziel definiert, einen Roboter zu konstruieren, der eigenständig Lieder auf einem Xylophon spielen kann. Dabei sollte der Roboter folgende Fähigkeiten besitzen:

1. **Autonomes Spielverhalten**: nach Kenntnis der Musiknoten den gesamten Spielprozess eigenständig durchführen können
2. **Lineare, bidirektionale Navigation**: sich mit einer konstanten Geschwindikeit gemäß dem vorliegenden Xylophon zu den richtigen Noten navigieren können
3. **„Musikalische" Regelungstechnik**: Bewegungen sowohl mechanisch als auch zeitlich präzise und geschickt steuern können für ein insgesamt musikalisch korrektes Spielen

Neben diesen angestrebten Eigenschaften haben wir uns außerdem folgende optionale Ziele gesetzt:

1. **Beidhändiges Spielen**: Benutzung von zwei Händen zum Spielen statt nur einer Hand
2. **Fortgeschrittene Koordination zweier Hände**: paralleles Zusammenspielen zweier Hände durch einen geeigneten Koordinierungsmechanismus
3. **Autonome Notenerkennung**: Selbstständige Erkennung von Musiknoten auf einem Blatt Papier mithilfe von Künstlicher Intelligenz

Durch das Erreichen dieser Ziele erhofften wir uns auch einen umfangreichen Einblick in das Gebiet der Robotik und v.a. wertvolle praktische Erfahrungen in dessen Entwicklungsprozsess. Dafür überlegten wir uns die Rollen Projekt-Managerin, Roboter-Ingenieur, Software-Entwickler und Musikerin, die jeweils für Dila, Si-Hoon, Batuhan bzw. Aneta eingeplant wurden.

Die geplanten Meilensteine umfassten den Bau einer Bewegungsplattform, die Konstruktion der Roboter-Hände, das Zusammenfügen von Hand und Bewegunsplattform, das Auslesen von den Noten durch einen Farbsensor, die Programmierung der Roboter-Logik, Verbesserungen für einen besseren Klang sowie das gleichzeitige Bewegen der beiden Hände. Jeder Meilenstein sollte eine verantwortliche Person sowie ein vorher geplantes Enddatum haben, begleitet von einer kurzen Beschreibung der Aufgaben und der damit verbundenen Probleme und Lösungsansätze.

## Tatsächlicher Ablauf

Jedoch wie so oft im Leben, lief das Projekt nicht exakt nach dem ursprünglich festgelegten Plan, sondern wurde im Laufe der Zeit immer agiler, sodass wir stets dynamisch auf derzeitig auftretende Probleme reagiert und diese gelöst haben. Auch die anfangs abgestimmte Rollenverteilung erwies sich als nicht unbedingt nützlich, sodass wir das Projekt grundsätzlich in zwei Teile untergliedert haben, den Hardware- und Software-Teil, woraus sich letztendlich die folgende Rollenverteilung gebildet hat:

- **Hardware-Konstruktion**: Dila, Si-Hoon, Aneta
- **Software-Entwicklung**: Si-Hoon, Batuhan

Trotz dieser Änderung von einem fest strukturierten zu einem sehr dynamischen Projektablauf haben wir uns Meilensteine gesetzt bzw. die ursprünglichen an die neuen Bedingungen angepasst, sodass wir uns während der Entwicklung immer wieder an eine allgemeine Struktur orientieren konnten. Unser erstes Ziel war es, den elementarsten Mechanismus unseres Xylobots zu realisieren: das tatsächliche Spielen einer Xylophon-Note mit einem Holzschlägel, erstmal für nur eine Hand. Für den nächsten Meilenstein sollte sich die Hand entlang des Xylophons zu einer beliebigen Note bewegen können, zunächst ohne die Geschwindigkeit zu berücksichtigen. Danach sollte für spätere Optimierungsmaßnahmen bzgl. dem Tempo eine zweite Hand hinzugefügt werden, die sich genau wie die erste Hand entlang des Xylophons bewegen kann. Mit dem Erreichen dieses Meilensteins sollte auch die Xylobot-Hardware zum größten Teil fertig gebaut sein. Bis hierhin war das zentrale Prinzip in unserem Entwicklungsprozess, sowohl in Hardware als auch in Software zwar parallel, aber dennoch gemeinsam voranzukommen, sodass Fortschritte und Änderungen in den Hardware-Komponenten immer auch mit der entsprechenden Software getestet werden können. So haben wir zuerst die Hardware für eine Hand gebaut, die dann auch direkt mit der Schlag-Funktion in der Software getestet wurde. Danach haben wir die Bewegungsplattform konstruiert, auf der wir auch die Hand-Bewegung mit der entsprechenden Bewegungsfunktion in der Software getestet haben. Anschließend, um eine zweite Hand einzuführen, wurde die zuvor gebaute Hand und Plattform repliziert, die dann zusammengesetzt und in das bisherige System integriert wurde. In all diesen Schritten wurde sowohl mit der Hardware- als auch der Software viel experimentiert, um die bestmögliche Umsetzung der Spiel-Bewegungen sowie des Xylophon-Klangs zu erreichen.

Alle weiteren Meilsteine waren hauptsächlich Software-Konzepte, mit denen wir ein zunehmend fortgeschritteneres Niveau in der Koordination der Hände erreichen wollten, was für die allgemeine Verbesserung des Xylophon-Spiels von zentraler Bedeutung ist. Dabei haben Batuhan und Si-Hoon die Software nach dem Top-Down-Prinzip und oft durch Pair-Programming entwickelt, was für die Projekt-Herangehensweise am sinnvollsten erschien. Wir haben uns also als Erstes überlegt, welche Aspekte des Xylobots durch die Software dargestellt werden sollten. Dazu wurden zunächst für alle wichtigen, d.h. sich tatsächlich mechanisch bewegenden Hardware-Komponenten sowie für die Musik-Logik eigene Klassen erstellt. Danach wurde deren Zusammenarbeit ausgiebig diskutiert, konzipiert und modelliert, indem die Schnittstellen zwischen den einzelnen Klassen klarer definiert und schon teilweise implementiert wurden. Dabei haben wir lange nach einer guten Lösung gesucht, um die Hände für das Spielen bestmöglich zu koordinieren. Schließlich haben wir dafür, nach dem Vorschlag unseres Betreuers Stefan, eine neue, nur auf logischer Ebene existierende Rolle eines Anführers eingeführt. Dieser soll durch die Software-Schnittstellen mit den physisch existierenden Xylobot-Händen interagieren und ihnen Anweisungen geben, um so das gesamte Vorspielen des Mustikstück zu koordinieren. Damit waren zumindest der konzeptionelle Aufbau der Xylobot-Software schon festgelegt. Die restliche Arbeit bestand darin, all die skizzierten Ideen, Klassen und Methoden tatsächlich fertig zu implementieren und sie mit Details sowie Optimierungen zu verbessern. Dabei haben wir stets unser Motto beibehalten und auch hier die Software stets zusammen mit der Xylobot-Hardware getestet. Nach vielen solchen Iterationen haben wir am Ende einen finalen Abschlusstest durchgeführt, bei dem wir unser Xylobot mit bekannten, populären Liedern getestet haben. Dieser verlief erfolgreich und damit erklärten wir unser Xylobot als fertig und hießen ihn mit einem Happy Birthday Song (den er selber gespielt hat) willkommen. Der Rest dieser Dokumentation soll nun die Funktionsweisen der Xylobot-Hardware und Software näher erläutern.

# 3

# Hardware

## Erste Ideen und Grunddesign

Als Erstes mussten wir herausfinden, wie der LEGO MINDSTORMS EV3-Motor konfiguriert werden muss, um die Schlagbewegung simulieren zu können. Dazu haben wir einfach Legosteine zu einem Griff zusammengebaut. Anschließend haben wir den Xylophon-Schlägel in den Griff eingesetzt. Mit einigen grundlegenden Java-Anweisungen konnten wir die Schlagbewegung erstellen.

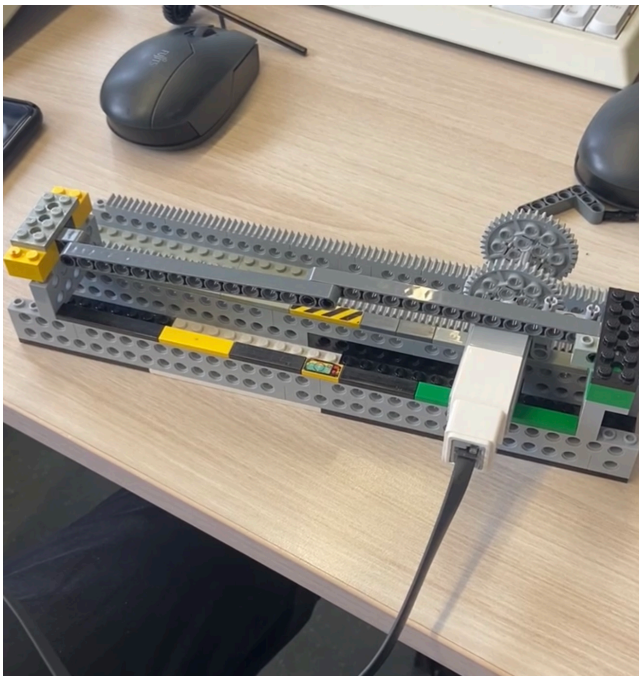


**Abbildung 3.1:** Erste Version der beweglichen Plattform und den Zahnräder

## Konstruktion der Bewegungsplattform

Aber wie wir im Projektplan bereits erklärt haben, bestand das Hauptziel darin, einen Roboter zu konstruieren, der wie ein Mensch Xylophon spielen kann. Der Roboter muss sich also entsprechend den gegebenen Anweisungen selbständig bewegen können. Um dies zu erreichen, mussten wir eine Bewegungsplattform bauen. Die Plattform besteht aus zwei Teilen: einer für die Bewegung der Zahnräder, wobei lange gezackte LEGO-Teile in zwei parallele Linien platziert wurden, auf denen sich die beiden verbundenen Zahnräder bewegten, und einer für die Bewegung des Motors, der das Rad antrieb. Hierfür wurden lange, glatte LEGO-Teile verwendet, damit der Motor mit dem Rad darauf gleiten konnte. Wir haben auch einige Teile parallel dazu hinzugefügt, um die gesamte Konstruktion zu stärken sowie weitere glatte Teile oben darauf, damit der Motor mit den beiden Zahnrädern nicht herausrutschte.

## Erste Probleme und Lösungen

Während dieser Phase stießen wir auf unser erstes Problem: Die Bewegung des Schlägels war nicht so präzise, wie wir uns es gewünscht hatten. Der Schlägel verharrte zu lange auf jeder Note, was zu längeren und weniger klaren Tönen führte. Wir dachten darüber nach, eine Sprungfeder unter den Schlägel zu platzieren, um die Töne knackiger zu gestalten. Am Ende entschieden wir uns jedoch dafür, die Bewegungsgeschwindigkeit der Hand zu erhöhen, um die Töne kurz und klar zu halten. Mehr dazu später im Software-Teil.

## Integration von Hand und Plattform

Später haben wir die Hand und die Plattform miteinander verbunden und alles, einschließlich des Xylophons, auf einer großen LEGO-Platte platziert, um alles an einem Ort zu halten. Der Roboter konnte nun selbstständig verschiedene Noten auf dem Xylophon spielen, allerdings noch ohne Kenntnis der Noten. Da es dem Roboter jedoch zu lange dauerte, zwischen den verschiedenen Noten zu wechseln, haben wir uns entschieden, einen zweiten Schlägel hinzuzufügen. Wir haben ihn auf derselben Plattform platziert, wobei jeder Schlägel in seiner eigenen Hälfte des Xylophons bewegt wurde. Dies war jedoch nicht optimal, da die Noten im höheren Spektrum seltener gespielt wurden, sodass wir nach einer anderen Lösung gesucht haben. Da wir keinen offiziellen Schlägel hatten, haben wir zwischen LEGO-Teilen, einem Bleistift und Lippenstiften gewechselt.

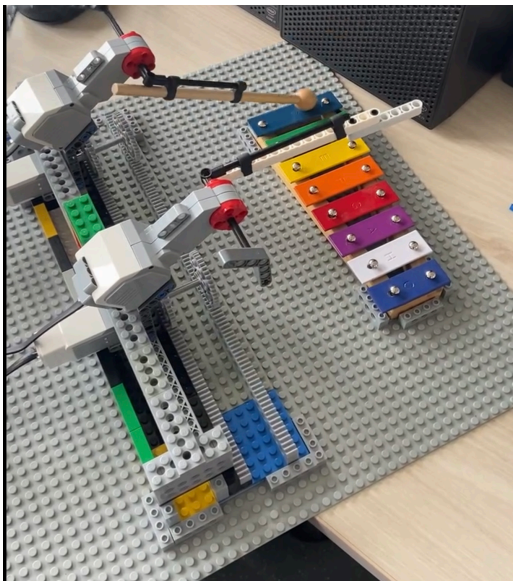


**Abbildung 3.2:** Beide Hände auf derselben Plattform

## Konstruktionsanpassungen und weitere Probleme

Ein weiteres Problem, das sich hierbei ergab, war die mangelnde Präzision in der Steuerung der Hand. Die Verwendung von LEGO-Teilen für die Konstruktion führte manchmal zu weniger stabilen Verbindungen zwischen den Teilen, was dazu führte, dass sich der Motor nicht reibungslos bewegen konnte. Wenn die LEGO-Teile nicht perfekt verbunden waren, stockte das Rad oder der Motor, der die Hand bewegte. Mit etwas Sorgfalt haben wir jedoch jedes kleine LEGO-Stück fest zusammengedrückt und versucht, den Kontakt der beweglichen Teile mit den problematischen Kanten zu minimieren.

## Bau einer zweiten Plattform

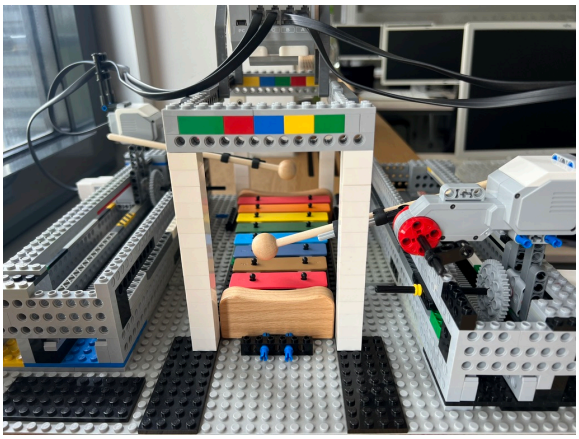
Wie bereits erwähnt, funktionierte das Beibehalten beider Hände auf derselben Plattform nicht so optimal, wie wir uns es gewünscht hatten, daher entstand der Bedarf, eine zweite Plattform zu bauen. Die zweite Plattform wurde genauso gebaut wie die erste: wir haben die Hand mit den Zahnrädern verbunden und alles auf der LEGO-Platte positioniert. Nun konnte sich theoretisch jeder der Schlägel über das gesamte Xylophon bewegen.

## Optimierungen und Verbesserung der Stabilität

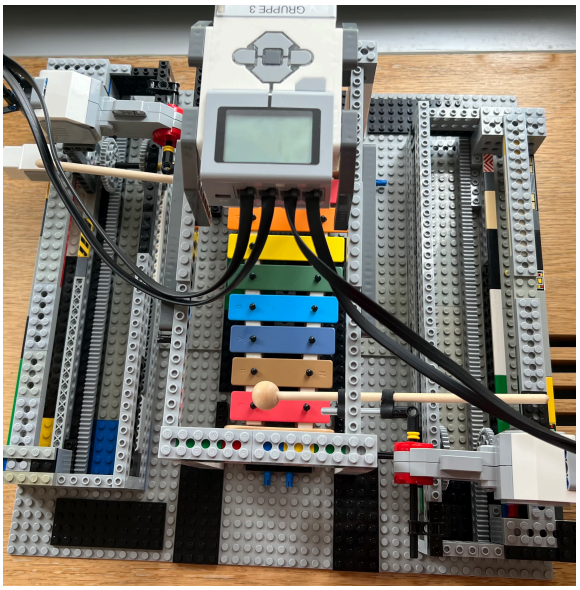
Da unser Roboter immer größer und komplizierter wurde, wurde er auch unübersichtlicher. Die vielen Motoren und Kabeln, die den Motor mit dem Akku verbanden, sorgten für ein chaotisches Erscheinungsbild. Zudem waren die Kabeln oft zu kurz und behinderten manchmal die Bewegung der Hände. Einige dieser Probleme verursachten sogar Zwischenfälle, bei denen sich die beweglichen Teile mit den Kabeln verfingen. Um dieses Problem zu lösen, haben wir ein kleines Dach über dem Xylophon gebaut, mit vier Säulen, auf denen der Akku platziert wurde. Nun gingen die Kabeln alle nach oben und behinderten die Bewegung der Hände nicht mehr.

## Austausch des Xylophons und Feineinstellungen

Zweitens haben wir uns entschieden, ein neues Xylophon zu kaufen. Die Töne des alten Xylophons waren nicht so klar, wie wir es uns gewünscht hatten, und die einzelnen Noten waren auch nicht breit genug. Die fehlende Klarheit der Töne beeinträchtigte die Qualität der Musikwiedergabe. Wir brauchten auch einen weiteren Schlägel, da die Töne mit dem Bleistift nicht so gut waren wie mit dem Original. Der neue Schlägel war besser geeignet, klare und präzise Töne zu erzeugen, was die Musikwiedergabe des Roboters deutlich verbesserte.



(**a**) Seitensicht



(**b**) Obensicht

**Abbildung 3.3:** Roboter aus verschiedenen Perspektiven

Wie wir in der Abbildung 3.3(a) sehen können, haben beide Hände unterschiedliche Höhen. Eine Hand befindet sich höher als die andere. Wir zielten hier darauf ab, Kollisionen der beiden Hände bei gleichzeitiger Bewegung zu verhindern, was öfters während der Entwicklungsphase zu einem Problem geführt hat. Aufgrund dieses Höhenunterschieds mussten wir für die Händen unterschiedliche Werte Schlagwinkel zuweisen. Dazu aber später mehr im Software-Teil.

# 4

# Software

## Allgemein

Die Software des Xylobots wurde mit der objektorientierten Programmiersprache Java realisiert. Dabei wurde das Open Source Java-Betriebssystem leJOS verwendet, das Konstrukte zur Steuerung des EV3-Bausteins bereitstellt. Zunächst soll die wesentliche Struktur bzw. Idee der dem Xylobot zugrunde liegenden Software dargelegt werden.

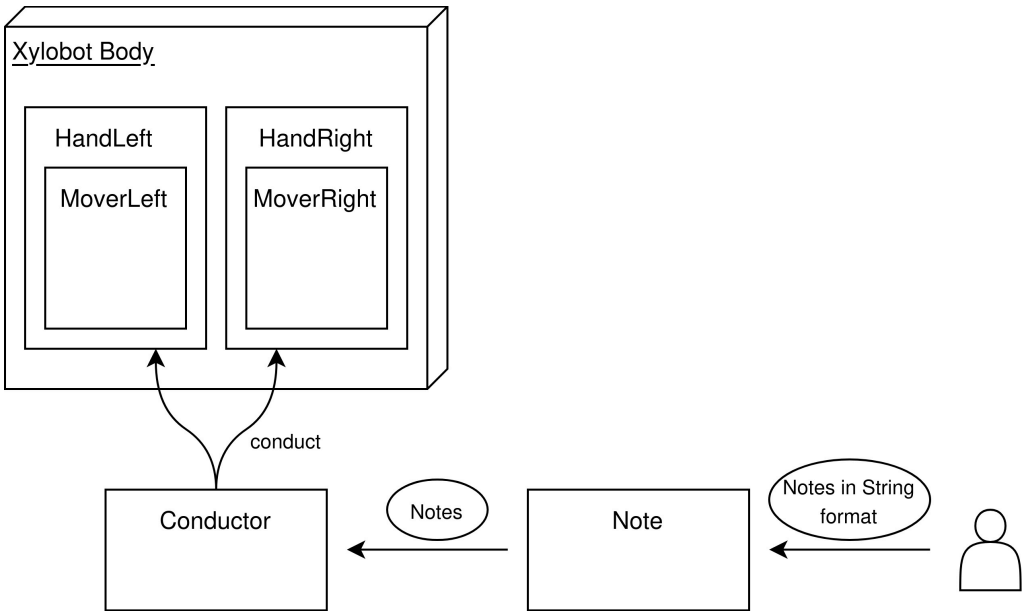



**Abbildung 4.1:** Illustratives Modell des Software-Aufbaus

Entsprechend der objektorientierten Natur von Java wurde dabei mit Objekten gearbeitet, die u.a. wichtige Hardware-Teile des Xylobots repräsentieren. Zur Veranschaulichung zeigt Fig. 4.1 ein illustratives Modell von der Software. In erster Linie besteht der Roboter aus dem Xylobot-Körper, der sich aus zwei Hand-Objekten zusammensetzt, die die beiden physischen Hände des Xylobots repräsentieren und jeweils in einem eigenen Thread laufen. Für jede Hand existiert außerdem jeweils ein Mover-Objekt, das die Hand auf der zugehörigen Schiene vor dem Xylophon bewegen kann. Neben der Implementierung des eigentlichen physischen Körpers existiert auf der Software-Ebene noch ein Conductor, also ein Dirigent, der im `main`-Thread die beiden Hände(-Threads) des Roboters untereinander koordiniert und so das Vorspielen des Musikstücks "dirigiert". D.h. es liegt ein gewisses Leader-Follower-Prinzip vor, bei dem der Conductor als Leader Anweisungen an die Hände übergibt, die dann diese als Follower befolgen und dementsprechend handeln. Schließlich wird noch eine Schnittstelle zum Benutzer benötigt. Dafür gibt der Benutzer die Musiknoten des vorzuspielenden Stücks in einem geeigneten String-Format in das Programm ein. Die Klasse `Note` übersetzt diese daraufhin in ein Format, das der Dirigent versteht und nutzt, um das gesamte Musikstück zu leiten.

Da das Ziel des Xylobots darin liegt, ein Song musikalisch korrekt auf dem Xylophon zu spielen, ist es nicht

nur wichtig, dass der Xylobot einfach nur läuft, sondern v.a. auch wann genau und wie schnell/ lange er läuft. Daher nehmen in der Programmierung des Xylobots die folgenden Probleme einen zentralen Platz ein: 1. das Tempo bzw. der Rhythmus (Wie schnell/lange? Wann?), 2. die Koordinierung (Wer spielt wann, was?) und 3. die Bewegung sowie das tatsächliche Spielen. Im Folgenden sollen anhand der einzelnen Schritte des tatsächlichen Ablaufs des Roboters demonstriert werden, wie diese Probleme gelöst wurden. Dabei handelt es sich bei den ersten drei Schritten um Vorarbeit, also Vorbereitungen für das tatsächliche Vorspielen, die vor der Laufzeit getroffen werden.

## 1. Noten-Bearbeitung

Der erste Schritt ist die Bearbeitung der vom Benutzer eingegebenen Noten. Um mit Musiknoten bequember arbeiten zu können, haben wir eine Enum-Klasse `Note` erstellt, deren Elemente musikalische Konstrukte verkörpern. Sie repräsentieren u.a. die 8 Noten auf dem Xylophon, wobei jedem ein Integer-Wert zugewiesen wird, der die entsprechende Position darstellen soll. Dieser wird benötigt, damit sich die Hände auf dem Xylophon orientieren können. Außerdem existieren noch weitere Elemente `REST(n)`, die den musikalischen Pausen entsprechen sollen, wobei `n` die relative Dauer der jeweiligen Pause zur Rhythmus-Einheit (siehe nächsten Abschnitt Anweisungen) ist.

**Note.java**

```
public enum Note {
    C(0), D(1), E(2), F(3), G(4), A(5), H(6), C1(7),
    REST1(1), REST2(2), REST3(3), REST4(4);
    ...
}
```

Da der Benutzer die Noten als Zeichenfolge eingibt, müssen diese noch in `Note`-Objekte umgewandelt werden, wofür die beiden Konvertierungsmethoden `charToNote()` und `strToNotes()` ins Spiel kommen. Dabei wandelt `charToNote()` mit dem Switch-Case-Konstrukt ein gültiges Zeichen in die entsprechende Note um, was von `strToNotes()` dazu genutzt wird, um durch alle Zeichen zu iterieren und diese in ein Array von Noten-Objekten zu konvertieren, womit unser Xylobot arbeiten kann:

**Note.java**

```
private static Note charToNote(char c) throws IllegalArgumentException {
    switch (c) {
    case 'c':
        return Note.C;
    case 'd':
        return Note.D;
    ...
    case 'C':
        return Note.C1;
    default:
        throw new IllegalArgumentException("Invalid note");
    }
}
```

**Note.java**

```
public static Note[] strToNotes(String notesStr) {
    Note[] notes = new Note[notesStr.length()];
    for (int i = 0; i < notesStr.length(); i++) {
        notes[i] = charToNote(notesStr.charAt(i));
    }
    return notes;
}
```

# 2. Anweisungen

## Vorgang

Nachdem nun die Noten zu einem richtigen Format umgewandelt wurden, werden im zweiten Schritt diese verwendet, um Spiel-Anweisungen bereits im Voraus an die Hände zu übergeben, damit diese beim tatsächlichen Spielen einfach nacheinander von ihnen befolgt werden können. Dafür ist die Methode `giveOutInstructions()` der `Conductor`-Klasse zuständig. In dieser analysiert der Dirigent die Musiknoten und weist jede Note genau einer Hand zu bzw. teilt ihr darüber mit. Die Kommunikation erfolgt dabei jeweils über eine Warteschlange, in die der Dirigent alle Anweisungen für die jeweilige Hand in der richtigen Reihenfolge einträgt, sodass die Hand später beim Vorspielen nacheinander die nächste Anweisung bzw. zu spielende Note daraus entnehmen kann, bis keine mehr übrig ist, d.h. alle ihr zugewiesenen Noten gespielt wurden. Dieser Zuweisungsvorgang ist illustrativ in der Abbildung 4.2 dargestellt.

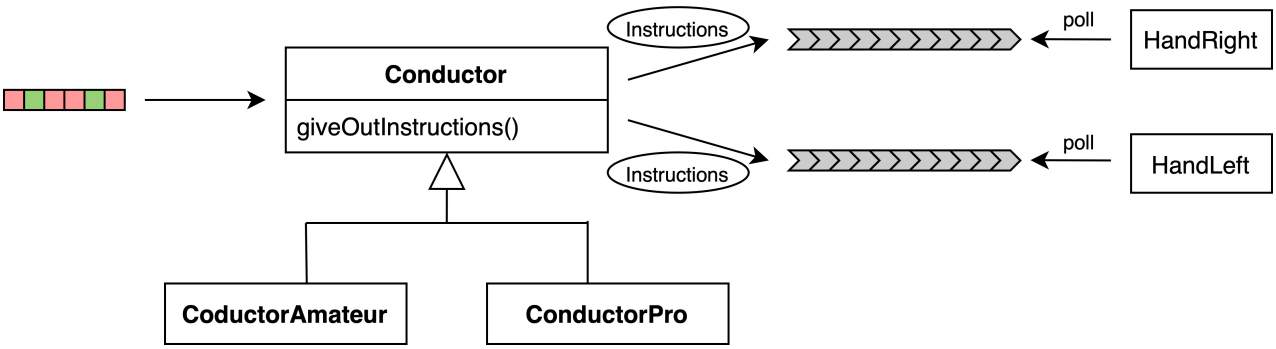



**Abbildung 4.2:** Illustratives Modell des Anweisungsvorgangs

Für die Warteschlange jeder Hand erzeugt der Dirigent ein `LinkedBlockingQueue`-Objekt, in das er wie oben beschrieben, die jeweiligen Anweisungen einträgt. Die Anweisung besteht dabei aus dem Motorumdrehungswinkel, den der Mover zurücklegen muss, damit die Hand sich anschließend in der richtigen Position für die nächste Note befindet. Dieser wird folgendermaßen ermittelt: die Methode `jumpsToNote()` gibt die Anzahl der Sprünge zurück, die die Hand von der jetzigen Position aus machen muss, um an der Zielnote anzukommen, wobei mit einem Sprung der physische Übergang von einer Xylophon-Note zu ihrer benachbarten Note gemeint ist. Aussßerdem gibt es noch eine Konstante der Klasse `Mover`, genannt `ANGLE_UNIT`, die den Motorumdrehungswinkel für genau einen physischen Noten-Sprung angibt. Dieser wurde experimental bestimmt (siehe Kapitel Bewegung) und wird also als eine Einheit interpretiert, die genau einen Sprung darstellt. D.h. der gesuchte Winkel ist das Produkt aus der Sprunganzahl und dem `ANGLE_UNIT`:

**Conductor.java**

```
BlockingQueue<Integer> handInstructions = new LinkedBlockingQueue<>();
...
public void giveOutInstructions(Note[] notes) {
    ...
    // number of jumps to be made from the current to the next note
    int jumpsNum = prevNote.jumpsToNote(destNote);
    // motor angle to be moved in order to arrive at the next note
    int angleToDest = jumpsNum * Mover.ANGLE_UNIT;
    ...
    handInstructions.put(angleToDest);
    ...
}
```

Die Grundidee hinter der Methode besteht also darin, durch jedes Noten-Objekt im Noten-Array zu iterieren und für jede Note zu bestimmen, welcher Hand sie zugewiesen werden soll und schließlich den basierend darauf berechneten Motorumdrehungswinkel der entsprechenden Hand mitzuteilen.

## Sprung-Flaschenhals

Es bleibt nun noch die Frage offen, wie man bestimmt, welche Hand was spielt. Zuvor sollte aber noch ein fundamentales Problem erläutert werden, das auftritt, wenn ein mechanischer Roboter Musikstücke rhythmisch

korrekt und auch in einem angemessenen Tempo spielen soll. Da unser Xylobot weder ein Zauberer noch ein reflex-fähiges Wesen ist, sondern auf mechanische Bauteile basiert, kann er nicht unmittelbar in einem schnellen Augenblick von einer Note zu einer anderen springen. Bis der `Mover` den Befehl zur Bewegung tatsächlich empfängt, der Motor gestartet wird und v.a. die Hand über die Schiene zu der richtigen Position bewegt wurde, braucht alles seine Zeit, die fundametal unvermeidlich ist. Es besteht also ein Flaschenhals, der je nach Musikstück bzw. Sprunggröße mehr oder weniger groß sein kann. Dies hat natürlich Einfluss auf den Rhythmus sowie das Tempo beim Vorspielen des Musikstücks. Man kann sich z.B. vorstellen, dass die Notensprünge von C zu D und C zu H nicht gleich schnell gespielt werden können, da sich die physischen Abstände in der jeweiligen Notenfolge stark voneinander unterscheiden. Damit trotzdem ein einheitlicher Rhythmus über das gesamte Musikstück hinweg gewährleistet wird, muss also die Zeit für den kleinsten Sprung genauso lang andauern wie der für den größten Sprung. Dafür haben wir eine Konstante, genannt `RHYTHM_UNIT`, eingeführt, die den Flaschenhals des größten Notensprungs abdeckt und so als Takteinheit einen einheitlichen Rhythmus vorgeben soll. Diese wird für jedes Musikstück individuell mit der Methode `evalRhythmUnit()` ermittelt, indem der größte in den Noten vorkommende Sprung berechnet und dessen experimental bestimmter Zeitwert in `RHYTHM_UNIT` eingesetzt wird. Wenn man nun alle Notensprünge zeitlich an diese Einheit anpasst, kann das gesamte Musikstück in einem einheitlichen Rhythmus gespielt werden.

## Anweisungsstrategien

Damit wäre zwar das Problem eines korrekten Rhythmus gelöst, jedoch hat alles keine Bedeutung, wenn der Xylobot dadurch Songs nur sehr langsam spielen kann. Deshalb besteht nun das Ziel darin, ein möglichst hohes Tempo zu erzielen, indem man (1) durch einen schlauen Dirigenten den größten Notensprung für jede Hand minimiert und/oder (2) die Anweisungen so gestaltet, dass die beiden Hände gut koordiniert parallel laufen können.

Der Standard-`Conductor` weist die eine 4-Noten-Hälfte des Xylophons der einen Hand und die andere Hälfte der anderen Hand zu. Dies ist eine sehr naive Variante, da weder die Notensprünge minimiert werden, noch viel Raum für Parallelität zwischen verschiedenen Notenstufen herrscht. Der Xylobot ist jedoch keinesfalls auf eine feste Anweisungsart bzw. Zuweisungsalgorithmus beschränkt. Sein Dirigent, demnach der Stil der Notenzuweisung kann durch weitere Dirigent-Klassen beliebig erweitert werden, indem man eine neue Klasse erstellt, die von der Standard `Conductor`-Klasse erbt und lediglich die `giveOutInstructions()`-Methode neu implementiert. Mit einer einzigen Zeilenänderung in der gesamten Codebasis kann dann der aktuelle Dirigent durch den neuen ersetzt werden (siehe die Klasse `Xylobot`). Unser Xylobot stellt z.B. noch `ConductorAmateur` und `ConductorPro` bereit, die das Vorspielen auf unterschiedlich fortgeschrittenem Niveau realisiert.

`ConductorAmateur` versucht die beiden oben genannten Ansätze umzusetzen, indem er alle zu spielenden Noten nacheinander immer abwechselnd den beiden Händen zuweist, es sei denn die nächste Note ist dieselbe wie die vorherige, oder eine unmittelbar danebenliegende Nachbar-Note. In dem Fall übernimmt dieselbe Hand, die auch die vorherige Note gespielt hat, die nächste. Dieser Ansatz scheint zunächst sinnvoll, da zum Einen durch das abwechselnde Spielen Parallelität geschaffen wird und zum Anderen der Sprung zwischen zwei nacheinder von der selben Hand gespielten Noten maximal eins beträgt. Jedoch hat sich herausgestellt, dass selbst eine optimale Minimierung des größten Notensprungs auf lediglich einen Sprung nicht zufrieden stellend ist, da der Flaschenhals trotzdem zu groß ist für ein schnelles Nacheinanderspielen der Noten.

Dieses Problem soll durch `ConductorPro` behoben werden, der unter etlichen Versuchen, einen brauchbaren Dirigenten zu entwickeln, das beste Ergebnis erreichte. Dabei ist seine Vorgehensweise überraschenderweise noch eine Stufe simpler als die von `ConductorAmateur`. Wie bei `ConductorAmateur` werden grundsätzlich alle Noten nacheinander abwechselnd den beiden Händen zugewiesen. Jedoch übernimmt jetzt eine Hand die nächste Note nur dann, wenn sie dieselbe ist wie die vorherige. Diese kleine Änderung in der Dirigierung ermöglicht ein deutlich schnelleres Zusammenspielen der beiden Hände als `Conductor` und `ConductorAmateur`. Denn dadurch dass sich die Hände in den allermeisten Fällen mit dem Spielen abwechseln, kann während die erste Hand die aktuelle Note spielt, die zweite Hand sich bereits von ihrer aktuellen Position zu der von der nächsten Note bewegen; sodass sie dann die Note direkt spielen kann, sobald die erste Hand fertig ist bzw. der Dirigent es so befiehlt. Der einzige auffällige Nachteil dieses Ansatzes ist, dass durch das starke Abwechseln die Gefahr besteht, dass sich die beiden Hände mit ihren Schlägeln ins Gehege kommen bzw. anstoßen (was beim Testen durchaus vorkam). Dagegen haben wir uns eine einfache Lösung überlegt, die im Hardware-Teil bereits erwähnt wurde: die Schlägeln in unterschiedlichen Höhen einstellen, was das Zusammenstoßen sehr unwahrscheinlich macht. Dies verspricht zwar keine festen Garantien, aber erwies sich durch viele Experimente und Tests statistisch gesehen als eine erfolgreich Lösung. Eine weitere Sorge dabei war jedoch, dass durch die unterschiedliche Weglänge beider Hände zum Xylophon, Inkonsistenzen bezüglich der Zeit und dem Ton beim Vorspielen aufterteten könnten. Letztendlich stellte sich aber heraus, dass die Sorge überflüssig war, da dieser hypothetische Unterschied beim tatsächlichen Spielen so gut wie nicht erkennbar ist.

## 3. Initialisierung

Nachdem die Hände nun über ihre Aufgaben Bescheid wissen, muss der Roboter nur noch in seinen Anfangszustand initialisiert werden, von dem er aus dann mit dem tatsächlichen Spielen beginnen kann. Dafür werden zunächst die Threads der beiden Hände gestartet, woraufhin die jeweiligen `Mover` angesprungen werden, die die Hände entlang der Schiene zu ihrer Startposition bewegen. Dies wird durch die `init()`-Methode der `Mover`-Klasse durchgeführt, indem ihre Motoren die Hände relativ zu ihrer Position solange nach links bewegen, bis sie eine LEGO-Wand erreichen, die sie zum Stillstand zwingt und so ihre Startposition markiert:

**Mover.java**

```
private RegulatedMotor motor;
...
public void init() {
    motor.setSpeed((int) (motor.getMaxSpeed()));

    // go to initial position
    moveLeft();
    while (true) {
        // stop when wall reached
        if (motor.isStalled()) {
            motor.stop();
            break;
        }
    }
}
```

Dadurch können wir die Position der Hände beim Start dynamisch festlegen, ohne dass wir sie bei jedem Durchlauf neu anpassen müssen. Aber nun stellt sich die Frage: Wie genau bewegen sich die Hände eigentlich entlang des Xylophons zu einer spezifischen Notenposition?

## 4. Bewegung

Für die Bewegung der Hände ist die `move()`-Methode verantwortlich, die die jeweiligen Motoren einfach um den übergebenen Winkel dreht; je nach Vorzeichen in die eine oder andere Drehrichtung. Das Wissen über ihre Umgebung, nämlich das Xylophon, vermitteln wir den Händen bzw. den Movern, indem wir alle Notenpositionen relativ zu ihrer jeweiligen Startposition setzen. Dafür kann der Integer-Wert der in der `Note`-Klasse definierten `Noten`-Objekte benutzt werden. Damit nun die Hände wissen, wie weit sie sich bei jedem Notensprung bewegen müssen, muss zunächst der nötige Motorumdrehungswinkel, für genau einen Notensprung ermittelt werden, d.h. der Übergang von einer Note zu ihrer direkten Nachbarsnote. Nach unseren Experimenten beträgt dieser Wert 69. Uns ist durchaus bewusst, dass dieser Wert nicht exakt sein kann und somit der kleine Fehler sich über Zeit vergrößern wird, was im schlimmsten Fall in einer falschen Position enden könnte. Nach unserem Test jedoch, der darin bestand, die Hände über eine längere Zeit von mehreren Minuten Notensprünge am Xylophon machen zu lassen, trat dieser Fall nie ein. Also solange das zu spielende Musikstück kein stundenlanges Orchester-Konzert sein soll, sollte diese Vorgehensweise kein Problem darstellen.

**Mover.java**

```
public static final int ANGLE_UNIT = -69;
...
public void move(int angle) {
    motor.rotate(angle);
}
```

Um nun beliebig große Notensprünge zu machen, berechnen wir wie im Abschnitt Anweisungen beschrieben den Sprungabstand zwischen der aktuellen Note und der Zielnote und multiplizieren ihn mit `ANGLE_UNIT`, sodass letztendlich der tatsächliche, physische Abstand abgelegt wird.

## Dirigierung

Endlich befinden sich nun beide Hände in ihrer jeweiligen Startposition und sind bereit mit dem tatsächlichen Spielen zu beginnen. Der Dirigent hatte in der Vorbereitungsphase die wichtige Aufgabe, den beiden Händen

ihre Aufgaben mitzuteilen. Hier kommt nun die zweite zentrale Aufgabe des Dirigenten zum Einsatz, nämlich die Leitung der tatsächlichen "Vorführung"des Xylophon-Songs, d.h. die Hände durch das gesamte Spiel anzuführen und angemessen zu koordinieren. Durch folgende in unserer Software gewährleisteten Bedingungen wird ein solches insgesamt musikalisch fundiertes Zusammenspielen ermöglicht:

(**1**) Der Dirigent besitzt ein Allwissen über die zu spielenden Noten und den dazwischen liegenden Pausen.

(**2**) Jede Hand kennt all die Noten, die von ihr nacheinander gespielt werden soll.

(**3**) Es besteht eine zuverlässige Kommunikation zwischen dem Dirigenten und den Händen, die eine globale zeitliche Synchronisation erlaubt.

Sowohl (1) als auch (2) sollten aus den Abschnitten Noten-Bearbeitung bzw. Anweisungen klar sein. Bei (3) ist zunächst mit zuverlässiger Kommunikation das interne Java-Feature der Thread-Signalisierung gemeint, bei dem durch ein gemeinsames Objekt Signale zwischen Threads gesendet bzw. auf sie gewartet werden kann. Dies ist gut geeignet für unseren Fall der Koordinierung nach dem Leader-Follower-Prinzip, da alle Rollen, sowohl Leader (Dirigent) als auch Follower (Hände) durch eigene Threads in Erscheinung treten. Bei der globalen zeitlichen Synchronisation sollten beide Aspekte, global und zeitlich, separat betrachtet werden. Die gloable Eigenschaft ist bereits durch die Verwendung des oben genannte Features erfüllt. Nämlich besteht eine logische Beziehung zwischen allen drei Rollen, in der sie sich ein gemeinsames Objekt teilen, durch das der Dirigent Signale an die beiden Hände schickt, um sie aufeinander abzustimmen.
Für die zeitliche Koordinierung jedoch, ist das Ganze nicht ausreichend, da sich noch der Zeitpunkt, zu dem die Signale gesendet werden, ausschließlich auf die Sicht des Dirigenten bezieht. D.h. es besteht die Möglichkeit, dass die Hand beim Empfang des Signals noch gar nicht zum Spielen bereit ist, weil sie z.B. noch nicht an ihrer nächsten Position angekommen ist. Ohne weitere Maßnahmen dagegen kann es also passieren, dass die Hand die Schlagbewegung ausführt, *während* sie sich noch in der Bewegung zur Note befindet. Um dies zu verhindern, haben wir im Code zusätzlich Java-Barrieren eingeführt, die garantieren sollen, dass stets das Spiel-Signal erst an die Hand gesendet wird, wenn sie auch in ihrer Position angekommen ist. Dafür wurde im Dirigenten-Thread sowie in den beiden Hände-Threads jeweils ein `CyclicBarrier`-Objekt an der richtigen Stelle eingesetzt. Diese Maßnahme sowie die Verwendung des Signal-Features sind in den folgenden Code-Ausschnitten zu erkennen:

**Conductor.java**

```
CyclicBarrier handReady = new CyclicBarrier(2);
...
public void conductPlay(Note[] notes) {
    ...
    handReady.await(); // wait for hand to arrive at its position
    synchronized (signalConductorToHand) {
        signalConductorToHand.notify(); // signal hand to play note
    }
    handReady.reset();
    ...
}
```

**Hand.java**

```
@Override
public void run() {
    ...
    mover.move(angleToDest);
    // arrived at the next position

    synchronized (signalConductorToHand) {
        handReady.await(); // ready to play note
        signalConductorToHand.wait(); // wait for signal from conductor
    }
    ...
}
```

Mit dieser zusätzlichen zeitlichen Synchronisation ist die Bedingung (3) somit ebenfalls erfüllt. Zusammen erlauben die drei Bedingungen das gewünschte Spiel-Verhalten: Der Dirigent kann alle Noten einzeln durch-

gehen und zum jeweils richtigen Zeitpunkt, d.h. nach eventuellen Pausen und wenn bereit, der eingeteilten Hand befehlen, die aktuelle Note tatsächlich zu spielen. Unmittelbar nach dem Spielen bewegt sich dann diese Hand zu ihrer nächsten Position und wartet auf das Eintreffen des Signals. Dieses Muster wird nacheinander für jede Note bzw. Hand solange wiederholt bis der Dirigent schließlich mit dem kompletten Musikstück fertig ist (siehe Abbildung 4.3).

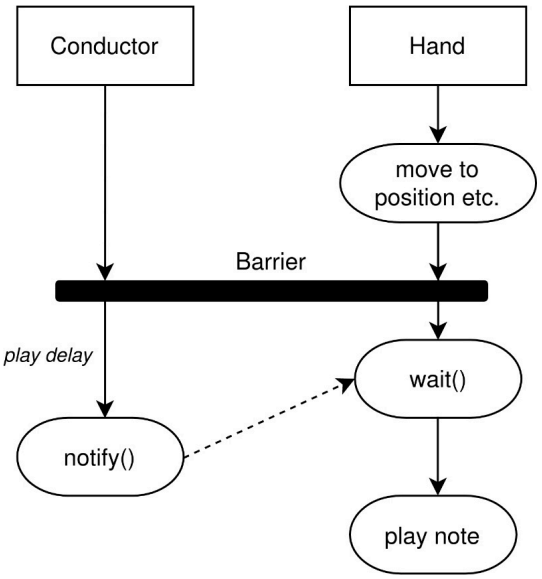



**Abbildung 4.3:** Illustratives Modell eines mögichen Vorgangs in der Dirigierung

## Spielen

Schließlich bleibt nur noch die finale Frage übrig, wie der Xylobot eine Note tatsächlich spielt. Die mechanische Schlagbewegung, bei der der Schlägel durch den Motor in der Hand nach unten bewegt wird, wird durch die Methode `hit()` ausgeführt:

**Hand.java**

```
public static int HIT_ANGLE_1 = 15;
public static int HIT_ANGLE_2 = 25;
...
public void hit(int hitAngle) {
    motor.rotate(hitAngle); // hit
    motor.rotate(-hitAngle); // reset position
}
```

Man erkennt direkt, sie funktioniert nach dem selben Prinzip wie die Bewegung für einen Notensprung. D.h. wir übergeben wieder den entsprechenden Winkel, diesmal um den Schlägel soweit nach unten zu bewegen, bis er die Xylophontaste trifft. Dieser ist für die beiden Hände verschieden, da wie in Anweisungen erwähnt die Hände sich in unterschiedlichen Höhen befinden. Dabei haben wir diese Winkel ebenfalls durch mehrere Experimente bestimmt, bei denen eine Schlagbewegung angestrebt wurde, die v.a. einen angenehm schönen Xylophon-Klang auslöst.

# 5

# Zusammenfassung

Mit der fertigen Software kann nun unser Xylobot 2.0 endlich zum Leben erwachen, Musik auf dem Xylophon spielen und somit Kindern zukünftiger Generationen ihren Traum erfüllen. Hurra! Zusammenfassend soll zum Schluss noch rekapituliert werden, welche von den gesetzten Zielen erreicht wurden und welche Grenzen sowie Verbesserungsaussichten für unser Roboter existieren.

Ein Kritikpunkt an unserem Xylobot wäre vielleicht, dass er keine außergewöhnlich schnellen Lieder spielen kann, was aber natürlich durch die Grenzen der gegebenen LEGO-Hardware durchaus Sinn ergibt. Wir argumentieren jedoch darüber hinaus, dass es zumindest konzeptionell keinen besseren Algorithmus als `ConductorPro` gibt. Denn jede andere Zuweisungsstrategie würde es theoretisch derselben Hand erlauben, zwei nacheinander zu spielende Noten zu übernehmen, was wie in Anweisungen erklärt zu einem Flaschenhals führen würde, der insgesamt nur ein langsames Spiel-Tempo ermöglicht. Eine Möglichkeit das Tempo zu verbessern, wäre es, mehr Hände zu verwenden. Dies würde aber auch mehr Plattformen benötigen, was bei einem kleinen Xylophon schwer skalierbar ist. Ein weitaus ernst zu nehmenderes Problem an unserem Xylobot ist, dass sich sein Xylophon nur über eine Notenoktave (+ 1 Note) erstreckt und auch keine schwarzen Tasten besitzt, was uns leider stark in der Auswahl an spielbaren Liedern limitiert. Die vermutlich einfachste Lösung dagegen wäre es ein größeres, umfangreicheres Xylophon zu benutzen, das mehr Noten besitzt, was dann aber natürlich eine Anpassung der physischen Hand-Plattform und auch Teile der Software erfordert.

Aber ansonsten konnten wir wie in unserem Projektplan definiert, die wichtigsten Ziele erreichen, sodass unser fertiger Xylobot letzten Endes alle angestrebten Fähigkeiten besitzt: er kann (1) nachdem die Musik vom Benutzer eingegeben wurde, autonom das gesamte Lied spielen, (2) sich entlang des Xylophons von Note zu Note navigieren und (3) seine Spiel-Bewegungen insgesamt musikalisch korrekt realisieren. Zudem haben wir auch zwei der drei optionalen Ziele erreicht: der Xylobot besitzt zwei unabhängige, parallel laufende Hände, die durch einen fortgeschrittenen Dirigenten, den `ConductorPro`, geschickt koordiniert werden. Somit sind wir insgesamt sehr zufrieden mit unsererem Projekt und dessen Endergebnis.

Das Ergebnis dieses Praktikums war jedoch nicht nur der fertige Xylobot 2.0. Durch die gemeinsame Gruppenarbeit an einem interessanten Projekt konnten wir außerdem wertvolle Erfahrungen im Team- sowie Projektanagement sammeln und dadurch unsere Teamfähigkeiten stark weiterentwickeln. Weiterhin haben wir durch die praktische Auseinandersetzung mit Robotern gelernt, dass ein korrektes Programm nicht direkt korrektes Verhalten im Roboter bedeutet. So wissen wir jetzt genau, dass bei der Entwicklung von realen System Hardware und Software nicht isoliert, sondern stets gemeinsam betrachtet werdenn muss, um Lösungen entwickeln zu können, die die Kluft zwischen den beiden Ebenen überbrücken. In der Summe konnten wir viele wichtige Einblicke in die Roboterentwicklung gewinnen, was unsere Problemslösungsfähigkeiten zweifellos geschärft hat.