Institut für Software-Entwicklung
Abteilung SE2
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3378

Konzeption eines Lego-Mindstorm Workshops für Studieninteressenten

Raimund Metzler

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Wagner
Betreuer:	Dipl.-Ing. Jan-Peter Ostberg
begonnen am:	25. September 2012
beendet am:	24. Januar 2013
CR-Klassifikation:	K.3.1

Kurzfassung

Im Rahmen dieser Diplomarbeit entstand ein Workshop, welcher das Interesse von Studieninteressenten an der *Softwaretechnik* wecken soll. Die Arbeit beschreibt wie und in welcher Form, LEGO-Mindstorms als interessante Basis des Workshops zum Einsatz kommt. Folgend wird beschrieben wie die Anforderung, die Mindstorm-Roboter mit Java zu programmieren, umgesetzt wurde und wie die Teilnehmer diese Aufgabe bewerkstelligen sollen. Die Diplomarbeit beschreibt des Weiteren den entstandenen Workshop, was er den Teilnehmern vermittelt und mit welchen Stilmitteln gearbeitet wird. Um den Workshop und dessen Teilnehmer zu unterstützen, entstanden beispielsweise einige Materialien wie eine Präsentation, Handouts und Beispiele, welche die Teilnehmer verwenden können.
Behandelt wird ebenfalls die Entwicklung der Roboter für den Workshop, welche Anforderungen diese zu erfüllen haben und beschreibt das Szenario, in welchem die Roboter zum Einsatz kommen. Da sich der Workshop um Elemente der Softwaretechnik dreht, werden auch diese Elemente erwähnt und ausgearbeitet.

Abstract

During the work on this diploma thesis, a workshop was created to make prospective students more interested in *Software-Engineering*. The thesis describes how LEGO-Mindstorms is used as an engaging component to work with. Furthermore it describes how Java, as a requirement of the thesis, was brought into the workshop and how the participants are affected by it. As the workshop was the main task of the thesis, it is described with all its steps, elements and materials used to support the participants. Since the workshop is about *Software-Engineering*, some aspects of it are part of the thesis as well as the workshop it self.
The diploma thesis also describes the robots and the requirements implied by scenario they are used in. Finally the software-suite the participants are using during the workshop is explained as well as the challenges and problems, which had to be solved.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Motivation

Die meisten Studieninteressenten sind nicht wirklich gut über den Inhalt eines Studienganges informiert und entscheiden sich oftmals aus dem Bauch heraus für einen Studiengang, der scheinbar etwas mit dem eigenen Interessengebiet und eventuell mit den individuellen Stärken zu tun hat. Zumeist handelt es sich dabei um bekannte und schon seit einigen Jahren angebotene Studiengänge, wie die Informatik. Neben diesen bekannten Studiengängen, gibt es aber natürlich noch weitere, leider oftmals unbekanntere oder auch neue Studiengänge, welche möglicherweise viel besser das Interessengebiet eines Studieninteressenten treffen. So gibt es die Möglichkeit, Softwaretechnik [4] anstatt Informatik zu studieren. Dies wäre zum Beispiel eine praxisorientiertere Ausbildung gegenüber der Informatik.
Der Studiengang der Softwaretechnik ist aber verhältnismäßig unbekannt und wird oftmals mit der Informatik gleichgesetzt. Letztere hat des Weiteren nach wie vor höhere Studienzahlen [5] und den allgemein höheren Bekanntheitsgrad.
Die Motivation hinter dieser Diplomarbeit ist es, die Softwaretechnik gegenüber der Informatik weiter zu betonen und ein generell höheres Interesse am Studiengang zu wecken. Auf lange Sicht ist ein deutlich erhöhter Bekanntheitsgrad erhofft, so dass sich jeder Studieninteressent bewusst für die Softwaretechnik statt der Informatik entscheiden kann, sollte er dies wollen.

1.2. Aufgabenstellung

Das Ziel dieser Diplomarbeit ist es, einen „Workshop“ zu erstellen, mit dessen Hilfe das Interesse am Studiengang „Softwaretechnik“ geweckt bzw. gefördert werden soll. Um die Zielsetzung zu erreichen, muss der Workshop interessant gestaltet, aber von Laien zu bewerkstelligen sein. Mit LEGO [8] Mindstorms [9] als Motivationsbasis und ein paar wenigen darauf basierenden Szenarien, liegt ein interessantes Konzept vor, welches den Teilnehmern Spaß machen, gleichzeitig aber auch etwas Wissen vermitteln soll.
Da Interesse am Studiengang der Softwaretechnik geweckt werden soll, liegen einige wichtige Aspekte des Software-Engineerings im Kern des Workshops, die den Teilnehmern vermittelt werden sollen. Da es zeitliche Grenzen gibt die eingehalten werden müssen, ist es im Workshop nicht mögliche diese Aspekte in aller Tiefe zu behandeln.

Gliederung

Nach diesem einleitenden Kapitel, gliedert sich die Diplomarbeit folgendermaßen:

Kapitel 2 – Softwaretechnische Grundlagen im Workshop: Ziel dieses Kapitels ist es, die für den Workshop nötigen Grundlagen der Softwaretechnik zu erläutern.

Kapitel 3 – Robotik In diesem Kapitel werden die Lego-Bausteine, -Sensoren und -Aktoren dargestellt und beschrieben.

Kapitel 4 – LeJOS Dieses Kapitel stellt eine kurze Beschreibung der LeJOS-API, -Werkzeuge dar.

Kapitel 5 – Workshop Das folgende Kapitel beschreibt den geplanten Ablauf und alle einzelnen Schritte des Workshops, sowie die bisherigen Ergebnisse und Auswertungen der Pilotworkshops.

Kapitel 6 – Die Softwarebausteine als Programmiergrundlage im Workshop Dieser Teil der Arbeit führt die einzelnen, im Laufe der Diplomarbeit entwickelten, Softwarebausteine für den Workshop auf.

Kapitel 7 – Szenarien Dieses Kapitel beschreibt die Entwicklung der verschiedenen Szenarien, nennt die Gründe warum Andere verworfen und welches Szenario am Ende in den Workshop übernommen wurde.

Kapitel 8 – Probleme, Schwierigkeiten und Herausforderungen Dieses Kapitel beschreibt die Probleme und Schwierigkeiten die es zu lösen, aber auch zu akzeptieren galt und stellt die Herausforderungen dar, welche hard- und softwareseitig gemeistert werden mussten.

Kapitel 9 – Zusammenfassung und Ausblick

2. Softwaretechnische Grundlagen im Workshop

Ziel dieses Kapitels ist es, die für den Workshop nötigen Grundlagen der Softwaretechnik zu erläutern.

2.1. Abgrenzung

Der zu dieser Diplomarbeit gehörige Workshop, soll einige Grundlagen der Softwaretechnik vermitteln. Mangels zur Verfügung stehender Zeit im Workshop, können dabei jedoch nur einige wenige Aspekte der Softwaretechnik behandelt und in den Workshop aufgenommen werden. Diese wenigen Aspekte können nur soweit behandelt werden, wie es im Rahmen des Workshops sinnvoll ist.

2.2. Behandelte Aspekte der Softwaretechnik

2.2.1. Analyse

Die Analyse steht ganz zu Beginn im Entwicklungsprozess einer Software, man spricht von der „Anforderungsanalyse“ . Sie dient dazu die Kundenwünsche genauestens zu analysieren und auf diesem Wege die Anforderungen zu erheben. Nach IEEE [3] gibt es mehrere Definitionen für den Begriff „Anforderungen“:

requirement A condition or capability needed by a user to solve a problem orachieve an objective.
(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
(3) A documented representation of a condition or capability as in (1) or (2).

IEEE Std. 610.12-1990

Im weiteren Verlauf der Arbeit wird nach Ludewig und Lichter [14, S. 357, 16.2.1] von der ersten Definition ausgegangen und der Begriff „user“ als „Klient“ definiert.
Auch die „Analyse“ wurde nach IEEE definiert:

requirements analysis (1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements.
(2) The process of studying and refining system, hardware, or software requirements.

IEEE Std. 610.12-1990

Auch hier wird im Weiteren wieder von Definition (1) ausgegangen. Dies bedeutet also, dass alle Anforderungen vom Klienten, beispielsweise Auftraggeber oder Kunde, kommen und in der Analyse erfasst und gesammelt werden. Einmal erfasst, stellen die Anforderungen die Basis der zu erstellenden Spezifikation dar.

Während der Erfassung der Anforderungen ist ein möglichst penibles Vorgehen angebracht. Jedes Detail ist zu erfassen damit gewährleistet ist, dass Klient und Softwareentwickler das selbe Verständnis unter der jeweiligen Anforderung haben.

Anwendung im Workshop

Die Teilnehmer des Workshops bekommen im Laufe der Kick-Off-Präsentation (5.3), nur eine grobe Vorstellung der Spielregeln für das später folgende Szenario vermittelt. Genau an diesem Punkt soll die Analyse ansetzen und den einzelnen Teams die Möglichkeit geben, die Spielregeln genauestens zu hinterfragen und eine möglichst optimale Strategie zu entwickeln. Dies bedeutet aber auch, dass einige Teams möglicherweise den ein oder anderen Vorteil erfahren, den sie durch geschickte Ausnutzung von Grauzonen im Regelwerk erhalten könnten. Auf diese Weise soll der Wettkampf im Szenario etwas betont und der Workshop interessanter gestaltet werden.

2.2.2. Spezifikation

Die Spezifikation dient als gemeinsame Kommunikationsbasis zwischen Auftraggeber und Auftragnehmer und definiert genau, wie alle weiteren Arbeiten umzusetzen sind. Auch für die Spezifikation gibt es eine Definition im IEEE-Glossar:

specification A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and, often, the procedures for determining whether these provisions have been satisfied. See also: formal specification; product specification; requirements specification.

IEEE Std. 610.12-1990

Auch an dieser Stelle sei wieder Ludewig und Lichter [14, S. 375] erwähnt, welche den Begriff „Anforderungsspezifkation“ wie folgt angeben: „Die Anforderungsspezifikation dokumentiert die wesentlichen Anforderungen an eine Software und ihre Schnittstellen, und zwar präzise, vollständig und überprüfbar.“

Die Spezifikation wird für gewöhnlich einem oder mehreren Reviews [14, S. 282] unterzogen, bei welchen der Klient oder Kunde anwesend ist. Anhand der Ergebnisse dieser Reviews ist zu entscheiden, ob die Spezifikation angenommen wird, oder nochmals überarbeitet werden muss.

Anwendung im Workshop

Die von den Teilnehmern im Laufe des Workshops anzufertigende Spezifikation, soll bei Weitem nicht so umfangreich und genau sein, wie es für die Entwicklung einer Software nötig wäre. Die Teilnehmer sollen aber ihr Vorgehen im Szenario planen, gegen die analysierten Spielregeln prüfen und ihre Ergebnisse schließlich niederschreiben. Inhalt der Spezifikation:

- Analysierte Regeln
- Gefundene Grauzonen im Regelwerk
- Mögliche Vorteile und / oder zu beachtende Dinge
- Vorgehen zum Erforschen des Spielfeldes unter Berücksichtigung der Verteilung und Größe der Suchpunkte sowie der Spielfeldabmessungen
- Falls Bestandteil des Szenarios, Punktewertigkeit der Bojen und daraus resultierendes Vorgehen

Den Teilnehmern steht für die Spezifikation eine einfache Vorlage (A.2.4) zur Verfügung, welche im Anhang zu finden ist.

3. Robotik

In diesem Kapitel werden die Lego-Bausteine, -Sensoren und -Aktoren dargestellt und beschrieben, sowie auf den im Workshop zum Einsatz kommenden Roboter eingegangen.

3.1. LEGO-Mindstorms

LEGO-Mindstorms [9] erweitert die bekannten LEGO-Standartbausteine um einige Sensoren und Aktoren sowie einen programmierbaren NXT-Baustein (3.2), welcher die eben genannten Sensoren und Aktoren anspricht und mit Energie versorgt.
Online, durch die große Community, als auch direkt von LEGO, sind viele Bauanleitungen für unterschiedliche Projekte verfügbar [9].

3.2. Der NXT-Baustein

Abbildung 3.1.: NXT-Baustein mit Sensoren und Aktoren.

Der „NXT-Baustein“ stellt das programmierbare Herzstück von Mindstorms dar. Programmiert wird er in der Standartausführung mit Hilfe einer von LEGO gestellten Software [10]. Falls nötig, lässt sich über die NXT-Software auch die aktuellste Firmwareversion für den NXT-Baustein installieren.
Die Programme für LEGO-Mindstorms werden in Form von Bausteinen grafisch zusammengesetzt und anschließend auf den NXT gespielt. Eine richtige Verkabelung der Sensoren und Aktoren vorausgesetzt, kann das Programm dann direkt ausgeführt werden.

Der NXT-Baustein hat zur Kommunikation mit dem PC zwei Schnittstellen, bei welchen es sich um eine USB- und eine Bluetooth-Schnittstelle handelt. Alle weiteren Anschlüsse sind für Sensoren, von welchen insgesamt maximal vier Stück gleichzeitig betrieben werden können, und Aktoren, maximal drei gleichzeitig, vorgesehen. Zu beachten ist aber, dass Sensoren und Aktoren die selben Anschlüsse und Kabel teilen, jedoch mit unterschiedlichen Spannungen versorgt werden. Es ist stets auf eine korrekte Verkabelung zu achten, um Schäden zu vermeiden.
Der Baustein hat unterhalb des USB-Anschlusses, im linken Loch, einen Reset-Knopf. Dieser kann beispielsweise mit einer aufgebogenen Büroklammer betätigt werden, um ein laufendes Programm abzubrechen.

3.2.1. Sensoren

Für Mindstorms ist eine Vielzahl an unterschiedlichen Sensoren verfügbar. Beschrieben werden hier aber nur die im Rahmen der Diplomarbeit verwendeten Sensoren.
Probleme, die im Zusammenhang mit der Sensorik auftreten, werden in Kapitel „Probleme, Schwierigkeiten und Herausforderungen“ genauer ausgeführt.

Tastsensor

Der Tastsensor ist der vermutlich einfachste Sensor des Systems und liefert binäre Werte zurück, um den momentanen Zustand (Taster gedrückt, Taster nicht gedrückt) zu beschreiben.

Helligkeitssensor

Der dem Baukasten beiliegende Helligkeitssensor eignet sich nur zur Unterscheidung von starken Kontrasten. So kann Hell von Dunkel, Schwarz von Weiß oder ein kräftiges Rot von einem blassem Blau unterschieden und über den zurückgelieferten Helligkeitswert auch begrenzt einer Farbe zugeordnet werden. Zur Erkennung der farbigen Bojen im Szenario reicht der Helligkeitssensor aber nicht aus. Daher wurde der optional angebotene Farbsensor nachbestellt.

Farbsensor

Der Farbsensor unterscheidet auf eine maximale Distanz von ca. 5 cm zuverlässig die Farben

- rot
- blau
- gelb

- grün
- schwarz
- weiß

und weniger zuverlässig, Farbwerte zwischen den eben genannten. Je weiter zwei Farben auf der Farbtabelle von einander entfernt sind, desto zuverlässiger können diese auch erkannt oder unterschieden werden. Blautöne werden beispielsweise nur sehr schlecht von einander unterschieden.

Ultraschallsensor

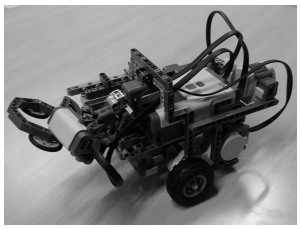
Der Ultraschallsensor ist der einzige, für LEGO-Mindstorms verfügbare Sensor, mit welchem auch über Distanzen von mehr als nur einigen Zentimetern Messungen durchgeführt werden können. Die maximale Reichweite ist mit (je nach Modell) ca. 170 cm angegeben, während bei weniger als 10 cm, Distanzen nur ungenau oder gar nicht gemessen werden können. Der Sensor arbeitet im Bereich von 20 cm aufwärts aber relativ genau und liefert, je nach Oberflächenbeschaffenheit, zentimetergenaue Ergebnisse zurück.

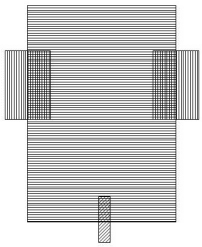
3.2.2. Aktoren

Dem Baukasten liegen genau drei Servomotoren bei, welche dem NXT-Baustein als Aktoren dienen. Die Maximalgeschwindigkeit hängt stark von Akku-Ladezustand oder den verwendeten Batterien ab, mit welchen die Aktoren für gewöhnlich mehr Leistung als mit dem mitgelieferten Akku erzielen.

3.3. Der Roboter des Workshops

Das Grundmodell des im Workshop zum Einsatz kommenden Roboters, basiert auf einem Bauplan, welcher unter „NXT-Programs“[16] unter dem Namen „Castor-Bot“[1] veröffentlicht wurde. Das Modell wurde so modifiziert, dass es aus den zur Verfügung stehenden Bauteilen umgesetzt werden kann. Da es sich aber nur um ein Basis-Modell zur Verwendung in weiteren Projekten handelt, wurde zusätzlich ein dritter Servomotor, ein Greifarm (3.3.1), ein Ultraschall-, ein Farb- sowie ein Tastsensor verbaut. Der Roboter hat drei Räder, wovon eines ein einfaches Stützrad ist, während die anderen beiden jeweils von einem Servomotor angetrieben werden.





3.3.1. Der Greifarm

Der am Roboter angebrachte Greifarm muss mit nur einem Servomotor, Bojen greifen und schließlich auch vom Boden heben können.

Abbildung 3.2.: Greifarm: Draufsicht und Schrägansicht

Das Zusammenspiel, der auf Zangen und Arm wirkenden Kräfte, ermöglicht dem Greifarm die nötigen Bewegungen.
Entwickelt wurde der Greifarm im Verlauf dieser Arbeit nachdem verschiedene Greifarme von Baggern und Kränen und deren Mechanik untersucht wurden. Vergleichbare Greifer aus Lego existieren, jedoch brachte die Recherche keinen Bauplan, der als Vorlage dienen konnte, zu Tage.
Der Greifer ist mit einem Farbsensor ausgestattet, um die Farbe einer angehobenen Boje bestimmen zu können.

3.4. Odometry

Bei der Odometry handelt es sich um die Positionsbestimmung eines sich fortbewegenden Objektes anhand von Daten des Antriebssystems. So kann beispielsweise die von einem immer gerade aus fahrendem Auto zurückgelegt Strecke berechnet werden, wenn der Raddurchmesser und die Zahl der Radumdrehungen bekannt sind. Die Fahrzeugrichtung kann über die Lenkwinkel der einzelnen Räder bestimmt werden.
Bezieht man die berechnete relative Positionsveränderung auf einen definierten Bezugspunkt, ist eine absolute Positionsbestimmung möglich.
Der Roboter aus dem Workshop hat keine unterschiedlichen Lenkwinkel, die Fahrzeugrichtung kann aber über den Radstand und die jeweilige Geschwindigkeit der beiden motorisierten Räder bestimmt werden. Zu beachten ist aber, dass äußere Einflüsse wie zum Beispiel durchdrehende Räder, die Positionsbestimmung verfälschen.
Den Nullpunkt des abgebildeten Koordinaten-Systems, stellt der Initialisierungspunkt des

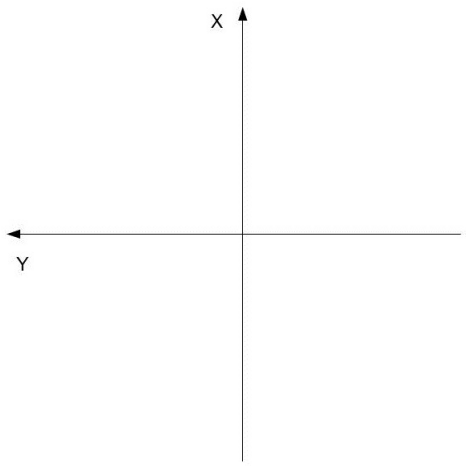


Abbildung 3.3.: Koordinatensystem der Odometry, Ausrichtung von X- und Y-Achse beachten!

Roboters dar. Im Workshop wird das immer die eigene Basis sein. Die Odometry des Roboters (siehe auch Kapitel „Pilot“) arbeitet mit Winkeln von 0° bis 180° nach links und -1° bis -179° nach rechts:

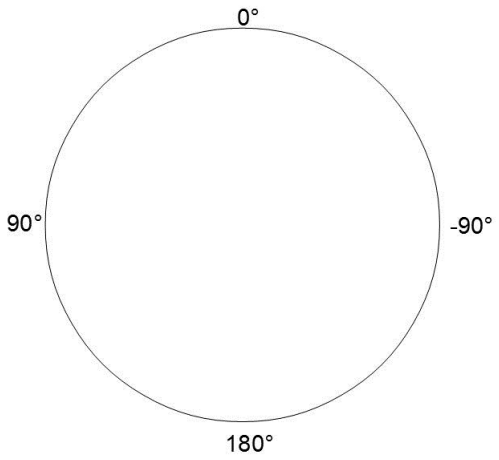


Abbildung 3.4.: Winkel der Odometry

4. LeJOS

Dieses Kapitel stellt LeJOS mit der zugehörigen Software und API vor.

4.1. LeJOS im Allgemeinen

Bei LeJOS [6] handelt es sich um eine kleine Java-VM, welche auf den NXT-Baustein gespielt werden kann. Dieser lässt sich dadurch mit Java und den dafür üblichen Programmierwerkzeugen programmieren. Das Projekt „LeJOS“ bietet außerdem die nötigen Treiber und Softwarekomponenten, um Java-Code auf den NXT-Baustein (3.2) zu überspielen. Unterstützt wird die Java-Entwicklung mit Hilfe der IDE Eclipse [7], für welche ein Plugin [12] existiert, mit dessen Unterstützung man den Code direkt aus der IDE auf den NXT-Baustein kopieren kann.

LeJOS ist ein quelloffenes Projekt und steht als solches unter der Mozilla-Public-License [15]. Das zum Projekt gehörende Forum ist die einzige Form eines Supportes. Allerdings erhält man diesen auch tatsächlich, sollte ein Problem oder eine Frage, welche durch das projekteigene Wiki [13] oder die Dokumentation [11] nicht beantwortet werden kann, auftreten.

LeJOS wird zwar schon seit einigen Jahren entwickelt, befindet sich aber noch immer in der beta-Phase des Projektes. Dies fällt hauptsächlich an kleineren Fehlern in der API, nicht implementiertem Funktionsumfang oder der an manchen Stellen unvollständigen Dokumentation auf.

4.2. Die LeJOS-API

Die von LeJOS zur Verfügung gestellte API [11] bietet prinzipiell den selben Funktionsumfang, wie die Standard Java-API, das Hauptaugenmerk liegt aber auf der Unterstützung der I/O-Ports des NXT-Baustein. Dinge wie Multi-Threading gehören aber dennoch zum Funktionsumfang, genauso wie Sockets oder die Möglichkeit in Dateien zu schreiben bzw. von diesen zu lesen. Jedoch ist anzumerken, dass der zur Verfügung stehende Speicherplatz auf dem NXT-Baustein mit 256kb sehr begrenzt ist.
Wie oben bereits kurz angesprochen, ist die API leider nicht ganz fehlerfrei. Während der Entwicklung der Softwarebausteine (6) für den Workshop, stellten diese Fehler immer wieder ein Problem dar, welches teilweise erst nach langem Debuggen als Fehler in der API erkannt

wurde.

5. Workshop

Das folgende Kapitel beschreibt den geplanten Ablauf und alle einzelnen Schritte des Workshops, sowie die bisherigen Ergebnisse und Auswertungen der Pilotworkshops. Des Weiteren wird vermittelt, auf welche Dinge man bei der Durchführung des Workshops achten muss.

5.1. Generelles zum Workshop

Die Zielgruppe des Workshops, sind Interessenten am Studiengang der Softwaretechnik. Der Workshop ist für maximal vier Teams mit jeweils höchstens drei Teilnehmern ausgelegt. Jedes Team bekommt dabei einen Roboter oder einen LEGO-Mindstorms [9] -Baukasten sowie einen Erweiterungskasten zur Verfügung gestellt, sollte der Roboter von den Teilenehmern selbst zusammengebaut werden.
Der Workshop ist für einen Zeitrahmen von ca. drei bis sechs Stunden ausgelegt. Laien ohne Hintergrundwissen werden diese Zeit auch benötigen, da doch ein gewisser Wissenstand nötig ist. Dieser wird aber, mit Hilfe einer ausführlichen Präsentation und zugehörigem Beispiel, vermittelt.
Wenigstens ein, besser zwei Betreuer, sollten den Workshop leiten und die Teilnehmer unterstützen, sollte dies von Nöten sein.

5.1.1. Die Bojen

Die Bojen für das Szenario müssen einige Bedingungen erfüllen. So waren die im Lego-Baukasten enthaltenen, etwa tischtennisballgroßen, roten und blauen Bälle ungeeignet und das nicht nur, weil sie auf Grund ihrer Form ständig über das Spielfeld rollen würden. Denn die Oberfläche einer Kugel reflektiert nur einen kleinen Teil der Ultraschallwellen zurück zum Sensor (siehe auch Kapitel „Verhalten von Ultraschallwellen“). Hinzu kommt die geringe Höhe, oder auch Radius der Kugel, aufgrund dessen die Kugel, je nach Sensorposition oder möglicher Neigung, nur sehr selten erkannt wurde.
Die jetzt für den Workshop zum Einsatz kommenden Bojen sind wesentlich höher und zylindrisch, was die Ultraschallwellen nur noch seitlich zerstreut, aber nicht mehr vertikal. Im Test wurde die aktuelle Boje selbst an den Rändern eines Suchkreises, also auf bis zu 50 cm Abstand, zuverlässig erkannt. Von ihrem Durchmesser sind die Bojen groß genug, um erkannt zu werden, aber klein genug, dass der Greifarm sie, selbst mit einem kleinen

Fehler in der Positionsbestimmung, noch gut greifen kann. Die Bojen sind mit einem kleinen Standfuß versehen, welcher es vereinfacht diese zu greifen, da sie nicht sofort umfallen, sollte der Greifarm gegen die Bojen stoßen.

5.1.2. Die Spielfeldgrenzen

Die Spielfeldabmessungen sind dem Roboter zwar bekannt, jedoch ist eine optische Abgrenzung dennoch anzuraten, um eine Grenzverletzung (Regelverstoß, siehe auch Kapitel „Regelwerk der Bojenjagd“) gleich erkennen zu können. Außerdem verhindert eine Spielfeldabgrenzung das Betreten des Spielfeldes durch Dritte.

5.2. Ablauf

5.2.1. Kick-Off

Der erste Schritt im Verlauf des Workshops, stellt eine einführende Präsentation (5.3) dar, welche den Teilnehmern den Ablauf und die Ziele des Workshops darstellen, als auch das nötige Wissen und die zugehörigen Grundlagen vermitteln soll. Die Präsentation ist ein äußerst wichtiges Element des Workshops, welches den weiteren Verlauf des Workshops stark beeinflussen kann. Je besser die Teilnehmer den präsentierten Stoff verstehen, desto wahrscheinlicher ist auch die erfolgreiche Anwendung des Selben im Workshop.

Benötigte Zeit: 30 bis 45 Minuten.

Das entstehende Beispiel zur Präsentation

Zur oben erwähnten Präsentation soll parallel zu dieser ein Beispielcode entstehen, welcher den Teilnehmern später als Vorlage dient. Je nach Gestaltung dieses Beispiels, kann der Schwierigkeitsgrad des Workshops stark variieren. Das Beispiel entsteht dabei entsprechend der behandelten Themen in der Präsentation. Der vorgestellte Stoff kann somit direkt vorgeführt werden. Die Teilnehmer können dabei in die Erstellung des Beispiels einbezogen werden, oder falls zuvor die passende Vorlage ausgeteilt wurde, auch direkt mitschreiben. Weitere Informationen sind unter Kapitel „Vorlagen und Beispiele“ zu finden.

5.2.2. Zusammenbauen des Roboters (optional)

Dieser Schritt ist, je nach zur Verfügung stehender Zeit, optional und kein Pflichtbestandteil des Workshops.
Die Teilnehmer sollen in ihren Teams (max. drei Teilnehmer = 1 Team) den für den Workshop benötigten Roboter, Schritt für Schritt aus den gestellten Lego-Bausteinen, zusammenbauen.

Für diesen Zweck steht eine Baueinleitung zur Verfügung, welche jeden einzelnen Schritt mit einer Abbildung darstellt.

Benötigte Zeit: 50 bis 80 Minuten (mehrere Teilnehmer können sich in Baugruppen aufteilen)

5.2.3. Analyse der Spielregeln

Im Laufe der Kick-Off-Präsentation werden die Spielregeln nur soweit erklärt, dass die Teilnehmer ungefähr wissen was sie zu tun haben. Die genauen Spielregeln sollen von den Teilnehmern im Laufe der Analyse selbst erkannt und begriffen werden. So wie in der für die Softwaretechnik typischen Analyse der Anforderungen im Gespräch mit dem Kunden, ist hierbei ein möglichst gründliches Vorgehen von Vorteil:
Eine nicht erkannte oder falsch analysierte Regel, welche im später folgenden Szenario gegen das tatsächliche Regelwerk (siehe auch Kapitel „Szenarien“) verstößt, führt zu einem Punkteabzug. Andererseits können Teams, welche das Regelwerk verstanden haben und dieses nach Möglichkeiten ausreizen, einen Vorteil gegenüber anderen Teams erlangen und Punkte durch Ausnutzung von Grauzonen leichter bzw. schneller erzielen.

Die Analyse unternimmt jedes Team getrennt von den anderen. Dies hat den Zweck, dass auch tatsächlich Vor- oder Nachteile erzielt werden können, sollten die Teams die Spielregeln unterschiedlich gut analysieren.

Benötigte Zeit: 5 bis 10 Minuten pro Team

5.2.4. Entwurf und Planung der Robotersteuerung

In der Entwurfs- und Planungsphase sollen alle Teilnehmer eine Strategie entwickeln, nach welcher der Roboter schließlich das Spielfeld erkunden soll. Die Teilnehmer sollen dabei durchaus zeichnerisch vorgehen und jeden einzelnen Schritt möglichst genau planen, um bei der Codierung Denkfehler zu vermeiden. Algorithmen können dabei auch in einer Programmablaufnotation (5.4) verfasst werden.
Die jeweilige Strategie eines Teams das Spielfeld zu erkunden, muss dabei im Vordergrund stehen: Die Pilotworkshops (5.6) ergaben wesentlich bessere und erfolgreichere Ergebnisse, wenn dieser Schritt vor allen anderen geplant wird und alles Weitere auf Basis dieser Grundlage erarbeitet wird.
Die Ergebnisse der Pilotworkshops als auch eigene, im Laufe der Diplomarbeit zu Testzwecken entstandene Robotersteuerungen, zeigten auf, dass unterschiedliche Strategien das Spielgeschehen im Szenario stark beeinflussen können. Stark überlappende Suchkreise finden in fast allen Fällen jede Boje auf dem Spielfeld, letzteres wird aber deutlich langsamer erkundet als bei weniger stark überlappenden Suchkreisen.

Um die Teilnehmer etwas zu Unterstützen ist es Sinnvoll, die im nächsten Schritt beschriebenen Zwischenziele (Suchpunkte anfahren, Bojen lokalisieren, Bojen greifen/absetzen) zu nennen, damit auch diese geplant und spezifiziert werden können.

Benötigte Zeit: 20 Minuten

5.2.5. Codierung

In diesem Schritt soll die tatsächliche Robotersteuerung implementiert werden. Als Grundlage dient dabei der in der vorhergehenden Phase entstandene Entwurf.
Gearbeitet wird mit Java und den in dieser Diplomarbeit entstandenen Softwarebausteinen (6). Das hierzu nötige Wissen erhalten die Teilnehmer aus der Kick-Off-Präsentation, möglichen Hand-Outs sowie der zur Verfügung stehenden Dokumentation der Bausteine.
Als Entwicklungsumgebung kommt „Eclipse“ [7] zum Einsatz, da es für diese IDE Unterstützung und AddOns direkt von LeJOS [6] gibt.

Bei der Codierung müssen die Teilnehmer Zwischenziele erreichen, welche einzeln getestet werden:

- Erfolgreiches Anfahren der einzelnen Suchpunkte.
- Lokalisieren einer Boje an einem Suchpunkt.
- Annähern und Greifen einer Boje und diese zur Basis bringen. Anschließende Rückkehr an den letzten oder anderweitig definierten (Such-)Punkt.

Die Zwischenziele sollen den Teilnehmern helfen, in kleinen Paketen zu denken. Ein Fehler / Bug während dem Testlauf für eines der Zwischenziele ist leichter zu identifizieren, wenn nicht der gesamte Code danach abgesucht werden muss.
Es macht Sinn, die oben gelisteten Zwischenziele schon in der Planungsphase zu berücksichtigen und zu nennen.

5.2.6. Antritt im Szenario

Im letzten Teil des Workshops müssen sich die Roboter der einzelnen Teams im Wettstreit gegeneinander beweisen. Spätestens in dieser Phase des Workshops, werden sich Vorteile, durch eine gründliche Analyse des Regelwerkes während der Befragung oder eine saubere Planung, aufzeigen und das Geschehen hoffentlich abwechslungsreich beeinflussen.
Die Einzelheiten und das dem Szenario zu Grunde liegende Regelwerk des Spiels, sind im Kapitel „Szenarien“ abgehandelt.

5.3. Einzelheiten zur Einleitung

Die Präsentation beschreibt Java-Sprachelemente, beginnend mit Variablen und für den Workshop relevante Variablen-Typen. Da der Workshop auch etwas Wissen über Klassen und Objekte voraussetzt, wird auch dieses Thema, soweit wie für die gestellten Aufgaben benötigt, behandelt. Den Abschluss bilden Java-Sprachelemente wie Methoden, Schleifen und if-/else-Konstrukte und natürlich die zugehörige Syntax.
Die Präsentation vermittelt den Teilnehmern das nötige Wissen zur Implementierung der Robotersteuerung und bringt allen Teams möglichst gleiche Chancen, die gesteckten Ziele zu erreichen und den Workshop erfolgreich zu absolvieren.
Da sich der in der Präsentation vermittelte Stoff, laut den Ergebnissen der Pilotworkshops (5.6), als sehr umfangreich entpuppte, wurde der Vortrag um „Live-Beispiele“ erweitert. Im Laufe des Vortrags soll auf diese Weise eine primitive, ineffektive aber durchaus lauffähige Robotersteuerung entstehen. Enthalten sind alle, in der Präsentation behandelten, Sprachelemente. Diese Robotersteuerung (A.2) kann den Teilnehmern durchaus zur Verfügung gestellt werden und als Vorlage dienen. Je nach Gestaltung dieser Vorlage, kann der Schwierigkeitsgrad des gesamten Workshops stark beeinflusst werden.

Bei gänzlich unerfahrenen Teilnehmern, würde sich ein kooperatives Vorgehen zur Erstellung des Beispiels anbieten. So würden die Teilnehmer Schritt für Schritt ihr eigenes Beispiel im Laufe der Präsentation erstellen. Zu beachten ist dann aber der zusätzliche Zeitaufwand für jeden Schritt des Beispiels. Da jedes Team die behandelte Zeile Code selbst verfassen muss und mögliche Fragen dazu stellen könnte, sind pro Schritt weitere 1-10 Minuten (je nach Umfang des zu verfassenden Codes) anzurechnen. Hinzukommen mögliche Fehler, die jedes Team bei der Codierung des Beispiels begehen könnte. Positiv wäre hingegen der Lerneffekt zu erwähnen. Das vermittelte Wissen aus der Präsentation kann direkt in der Praxis angewandt werden und das allgemeine Verständnis fördern.
Sollte statt dem entstehenden Beispiel, eine der verfügbaren beiden Vorlagen (A.2.2, A.2.3) zur Verfügung gestellt werden, sind mögliche Fehler durch die Teilnehmer beim Mitschrieb ausgeschlossen, der Praxisbezug würde aber fehlen.
Ob das Beispiel jetzt kooperativ oder nicht entsteht, hängt stark von der zur Verfügung stehen Zeit ab. Bei zeitlich knapp bemessen Workshops wäre ein kooperatives Beispiel vermutlich zu aufwendig. Jedoch wäre eine mögliche Zeitersparnis bei der Planungs- und Codierungsphase möglich, da schon viel Code vorgegeben wurde.

Generell ist natürlich eine möglichst lebhafte Gestaltung des Vortrags anzuraten und das Publikum, gerade während der Erstellung des Beispiels, in das Geschehen mit einzubeziehen. Lohnenswert in den Vortrag investierte Zeit zahlt sich aus, da sich Unverständnis und Probleme besonders in der Codierungsphase zeitraubend bemerkbar machen.

Weitere Informationen zu den im obrigen Abschnitt genannten Vorlagen und Beispielen sind unter Kapitel „Vorlagen und Beispiele“ zu finden.

5.3.1. Vorlagen und Beispiele

Um die entstehende Robotersteuerung schon von Anfang an in einer gewünschten Art und Weise zu beeinflussen, stehen ein paar wenige Vorlagen für diese zur Verfügung. Je nach Vorwissen der Teilnehmer und gewünschtem Schwierigkeitsgrad des Workshops, stellen die Vorlagen schon mehr oder eben weniger komplettierte Robotersteuerungen zur Verfügung.

Einzelheiten zu Vorlage 1

Die erste, der beiden Vorlagen enthält nur einige Kommentarköpfe, eine fertig initialisierte Variable für den späteren Gebrauch, sowie ein komplett instantiiertes Objekt mit allen dazu gehörigen Parametern. Alle weiteren nötigen Variablen, Objekte, Schleifen und sonstige Sprachelemente, sind von den Teilnehmern selbst und auf Basis der einführenden Präsentation zu verfassen. Wie in „Ergebnisse der Pilotworkshops“ nachzulesen, gestaltet sich die Planungs- und Codierungsphase als durchaus anspruchsvoll. Sollte ein Workshop mit dieser Vorlage durchgeführt werden, ist mit vielen Fragen seitens der Teilnehmer zu rechnen.
Die hier beschriebene Vorlage ist im „Anhang“ unter „Vorlage 1 der Steuerung“ zu finden.

Einzelheiten zu Vorlage 2

Schon weitaus vollständiger und umfangreicher als die Erste, gestaltet sich die Zweite, der beiden Vorlagen. Diese stellt eine vollständige, aber noch sehr triviale und nicht optimierte Robotersteuerung zur Verfügung. Beinhaltet sind neben vielen Kommentarköpfen, die jede Einzelheit beschreiben, auch alle Sprachelemente der Präsentation. Die Teilnehmer müssen sich weniger mit der Definition und Instantiierung von Variablen, Objekten, Schleifen und Methoden auseinander setzen, als viel mehr mit deren richtiger Anwendung:
Die Robotersteuerung der Vorlage deckt nur einen sehr kleinen Teil des Spielfeldes (ein einzelner Suchpunkt) ab und würde eine Boje nur genau dann lokalisieren können, wenn sich diese innerhalb des Suchfeldes befinden würde. Die Teilnehmer haben die Steuerung also so anzupassen, dass Suchpunkte auf dem ganzen Feld verteilt, angefahren und untersucht werden. Des Weiteren sind die nötigen Vorgänge um Punkte zu erzielen, nur in durch Kommentare beschriebener Form vorhanden. Der Aufruf dieser Anweisungen an der passenden stelle, ist ebenfalls selbst zu bestimmen.
Vorlage 2 (A.2.3) bietet den Workshopteilnehmern ungleich mehr Unterstützung als Nr. 1, unterfordert aber möglicherweise erfahrenere Teilnehmer.

5.3.2. Anmerkung zur weiteren Entwicklung der Vorlagen

Die zur Verfügung stehenden Vorlagen sollten im Laufe von mehreren Workshops überarbeitet und so angepasst werden, dass Sie für den Großteil der Teilnehmer verwendet werden

können. Zu beachten ist dabei die tatsächlich gewählte Zeitvorgabe für den Workshop sowie die gewünschten Spielelemente im Szenario. Beispielsweise könnte man die unterschiedlichen Farben der Bojen missachten und jeder Boje die gleiche Punktezahl zuordnen, um den Schwierigkeitsgrad weiter zu vereinfachen. Im Falle der zweiten Vorlage, wäre der Schwierigkeitsgrad wohl sehr gering, da , neben dem Greifen der Bojen, nur noch Suchpunkte auf dem Spielfeld zu verteilen wären.

Das Beispiel aus der Präsentation

Wie schon erwähnt, entsteht im Laufe der Präsentation eine lauffähige Robotersteuerung, welche als Beispiel dienen soll. Ob diese Steuerung den Teilnehmern zur Verfügung gestellt werden soll oder nicht, hängt vom Leiter des Workshops ab. Beachten sollte dieser, wie Umfangreich das Beispiel gestaltet ist. Die im Anhang vorzufindende Beispielsteuerung (A.2.1) beinhaltet alle, in der Präsentation vorgestellten, Sprachelemente und verbindet diese zu einer komplett funktionstüchtigen Robotersteuerung. Mit einigen geschickt gesetzten Suchpunkten, würde diese Steuerung bereits Erfolge im Szenario erzielen und daher eine eher ungeeignete, da zu einfache, Vorlage für die Teilnehmer darstellen. Als Beispiel hingegen ist sie gut geeignet und könnte als Anregung für weitere Präsentationen hilfreich sein.

5.4. Details zur Planung der Robotersteuerung

Um zu verhindern, dass die Teilnehmer gleich nach der Präsentation beginnen ihre Robotersteuerung zu kodieren ohne sich über ihr Vorgehen im Klaren zu sein, sollte die Planungsphase explizit erwähnt und betont werden. Die Pilotworkshops zeigten auf, das unterschiedliche Teams zu unterschiedlichen Strategien neigen, das Spielfeld abzudecken.
Sollte eine Tafel oder ähnliches zur Verfügung stehen, kann durchaus ein kurze Liste über ein paar wenige zu beachtende Details aufgeschrieben werden. Zu beachten ist, dass man nicht jedes Detail nennt, um die Spannung im Wettkampf nicht negativ zu beeinflussen. Es geht vielmehr darum, Denkanstöße zu geben und die Teilnehmer zum Nachdenken zu animieren.
Zur Planung des weiteren Vorgehens ist eine kleine Vorlage (A.2.4) im Anhang zu finden. Die Vorlage ist sehr einfach gehalten (Titel, Namensfelder für Teilnehmer, vorgefertigte Zeilen), bewirkt aber allein durch ihre Existenz, dass die Teilnehmer ihr mehr Aufmerksamkeit schenken, als einem leeren Blatt Papier.

Programm-Ablauf-Notation

Programmablaufpläne stellen eine Möglichkeit dar, Programmkonstrukte graphisch darzustellen. So lässt sich vorhandener Code leichter verständlich präsentieren oder zu implementierende Programmkonstrukte besser Planen.
Die Programmablaufnotation könnte den Teilnehmern in der Planungsphase behilflich sein,

die einzelnen Schritte ihrer Vorgehensstrategie übersichtlich zu planen. Ein in dieser Notation verfasstes Programmkonstrukt, lässt sich oftmals leichter durchschauen, als ein nur durch Text erklärtes.
Zu beachten ist aber, dass Programmablaufpläne für große Programmkonstrukte auch sehr verwirrend sein können. Im Rahmen dieses Workshops sollte ein solcher Fall aber nicht eintreten. Die folgende Abbildung stellt die einzelnen Notationen dar:

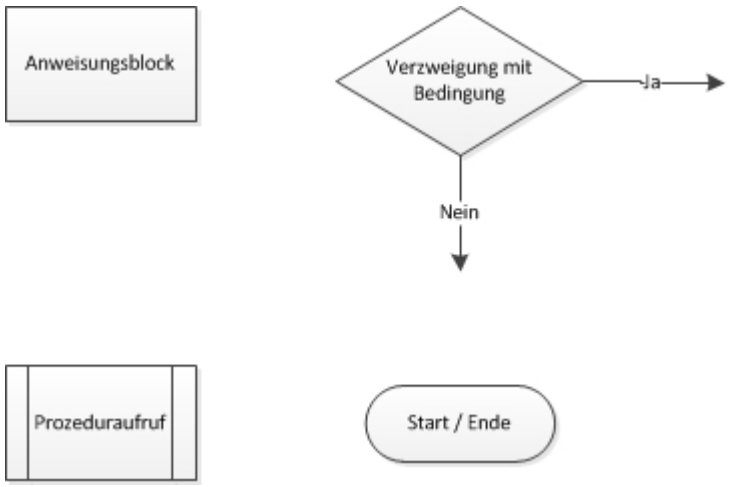


Abbildung 5.1.: Notation eines Programmablauf-Planes

5.5. Einzelheiten zur Analyse der Spielregeln

Die Spielregeln sind bewusst vage zu nennen und sollen Fragen aufwerfen. Tritt dieser Fall ein, herrschen optimale Bedingungen für die Analyse. Selbige ist, zugunsten des Wettkampfes, für jedes Team einzeln zu halten und am geschicktesten während der Aufbauphase des Roboters (falls Teil des Workshops) untergebracht. Sollte der Roboter fertig aufgebaut zur Verfügung gestellt werden (Zeitersparnis), sollte die Analyse möglichst zu Beginn oder noch vor der Planungsphase durchgeführt werden. Dies wäre zwar nicht optimal, da manche für die Planung nötigen Details, durch die Analyse beeinflusst werden könnten, jedoch lässt sich ohne den Aufbau des Roboters kaum ein anderer Zeitpunkt finden.

5.6. Ergebnisse der Pilotworkshops

Dieses Kapitel schildert die ausgewerteten Ergebnisse der Pilotworkshops.

5.6.1. Viel vorausgesetztes Wissen

Die Pilotworkshops zeigten einige Schwierigkeiten auf, das nötige Wissen an die Teilnehmer zu vermitteln. Die Zielgruppe stellen Studieninteressenten wie beispielsweise Abiturienten dar, bei welchen man aber kein umfangreiches Vorwissen voraussetzen kann. So hatten Testkandidaten ohne Vorwissen große Probleme der gesamten Präsentation zu folgen oder das daraus resultierende Wissen anzuwenden. Da aber das nötige Wissen für den Workshop nicht reduziert werden kann, ist ein besserer Weg nötig, das Wissen zu vermitteln.
Da die meisten Menschen durch praktische Anwendung, zuvor vermitteltes Wissen, besser behalten als trockene Theorie, soll im Laufe der Präsentation, in Zusammenarbeit mit dem Publikum, eine einfache aber ineffektive Robotersteuerung (A.2.1) entstehen. Diese dient nicht nur als Beispiel für die Präsentation, sondern später auch als mögliche Vorlage für die Workshop-Teilnehmer.
Die Vorlage ist so gestaltet, dass ohne signifikante Änderungen, keine erfolgreiche Teilenahme am Szenario des Workshops möglich ist.

5.6.2. Planungsphase kam zu kurz

Alle Teilnehmer der Pilotworkshops nahmen sich nur wenig Zeit, die Elemente, Algorithmen und nötigen Schritte durchzuplanen. Einige Testkandidaten ignorierten in der anschließenden Implementierung ihre Planung oder Spezifikation sogar komplett und gingen völlig anders vor. Die daraus resultierende Robotersteuerung zeigte im abschließenden Szenario aber nur wenig Erfolg. Testteilnehmer mit guter Planung hingegen, hatten meist sichtlichen Erfolg und konnten Punkte im Wettkampf für das eigene Team ergattern.
Um der Spezifikation etwas mehr Aufmerksamkeit zukommen zu lassen, ist eine Vorlage angedacht, welche von den Teilnehmern nur noch ausgefüllt werden muss. Eine ausgedruckte Vorlage zieht mehr Aufmerksamkeit auf sich, als ein leeres Blatt Papier, welches die Teilnehmer vielleicht sogar noch selber mitbringen müssen.

5.6.3. Benötigte Zeit

Testteilnehmer ohne Vorwissen benötigten ungleich mehr Zeit als solche, die schon einmal die eine oder andere Zeile Code verfassten, oder gar Erfahrungen in Java und Eclipse aufweisen können. Erfahrenere Teilnehmer konnten den Workshop in angemessener Zeit absolvieren, während Laien den Zeitrahmen sprengten.
Auch hier soll das oben beschrieben „Live-Beispiel“ während der Präsentation Abhilfe schaffen.

Anmerkung: Alle Protokolle der Pilotworkshops sind im Anhang (A) einzusehen

6. Die Softwarebausteine als Programmiergrundlage im Workshop

Dieses Kapitel beschreibt die einzelnen, in dieser Diplomarbeit entwickelten, Softwarebausteine für den Workshop. Dabei erreicht es nicht den Umfang einer kompletten Spezifikation, stellt aber den Anspruch, jeden Baustein ausreichend genau zu beschreiben sowie den Gedanken dahinter zu vermitteln.

6.1. Bausteine für den Workshop

Die Möglichkeit LEGO-Mindstorms-Roboter mit Java zu programmieren, ist eine interessante, für Laien aber auch sehr anspruchsvolle Beschäftigung. Um dies für Laien ohne Hintergrundwissen umsetzbar zu gestalten, galt es die Ansprüche an das nötige Know-How zu reduzieren. Im Laufe der Diplomarbeit entstanden, mit diesem Hintergedanken, Softwarebausteine, welche auch ohne Fachwissen eingesetzt werden können, um die gesteckten Ziele des Workshops zu erreichen. Im Laufe des besagten Workshops sollen die Teilnehmer eine Robotersteuerung, mit Hilfe dieser Softwarebausteine, entwerfen und umsetzen. Jeder Softwarebaustein kapselt dabei aufwendige Algorithmen oder generell komplexe Vorgänge, in leicht gebräuchliche und zueinander passende Klassen bzw. Java-Pakete. Mit Hilfe der Bausteine, können die Teilnehmer auf abstrahierter und stark vereinfachter Ebene, ihre Robotersteuerung implementieren. Da für den Workshop nur begrenzt Zeit verfügbar ist, ist dieser Punkt besonders wichtig, um den Teilnehmern nach Absolvierung des Workshops ein Erfolgserlebnis zu ermöglichen.

Manche Bausteine sind in zwei Ausführungen vorhanden: Eine Basisausführung und eine abstraktere Version, welche einige Vorgänge vereinfacht und dem Benutzer Arbeit abnimmt. Zweck der vereinfachten Versionen ist eine Zeitersparnis, welche bei zeitlich stark eingeschränkten Workshops oder Teilnehmern ohne Vorwissen zum Einsatz kommen kann.
Im Folgenden wird jeder Baustein vorgestellt und beschrieben.

6.2. ObjectInfo

Die Klasse „ObjectInfo“ speichert Informationen über vom Roboter bzw. Sensor gefundene Objekte auf dem Spielfeld. Neben mehreren Gettern und Settern beinhaltet ObjectInfo auch

eine Methode, die anhand der zur Verfügung stehenden Informationen entscheidet, ob es sich um eine relevante Boje, eine des gegnerischen Teams, oder ob es sich überhaupt um eine Boje handelt.
Die Abfrage gestaltet sich für den Benutzer einfach, da nur ein bool'scher Wert berücksichtigt werden muss. Die Workshopteilnehmer, bzw. das entsprechende Team, geben die Team-Zugehörigkeit des Roboters an und erhalten zu jeder Boje die nötigen Informationen, um entscheiden zu können, ob diese viele Punkte wert ist oder nicht.

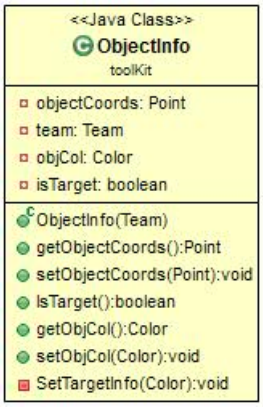


Abbildung 6.1.: Klassen-Diagramm: ObjectInfo

6.3. ObjectHandler

Der „ObjectHandler" verwaltet alle vom Sensor auf dem Spielfeld gefundenen Objekte und bietet dem Benutzer Möglichkeiten zum Einsehen, Hinzufügen oder Löschen der selben. Objekte werden als ObjectInfo (6.2) gespeichert. Der „ObjectHandler" speichert eine weitere Instanz von ObjectInfo nur genau dann, wenn diese, oder eine Instanz mit sehr ähnlichen Koordinaten, nicht bereits vorhanden ist. Die Folgende Abbildung stellt symbolisch 3 Objekte dar, von welchen „Object 1" bereits gespeichert wurde, während „Object 2" und „Object 3" hinzugefügt werden sollen. Da „Object 2" aber ähnliche Orts-Koordination wie „Object 1" hat, wird es nicht gespeichert.

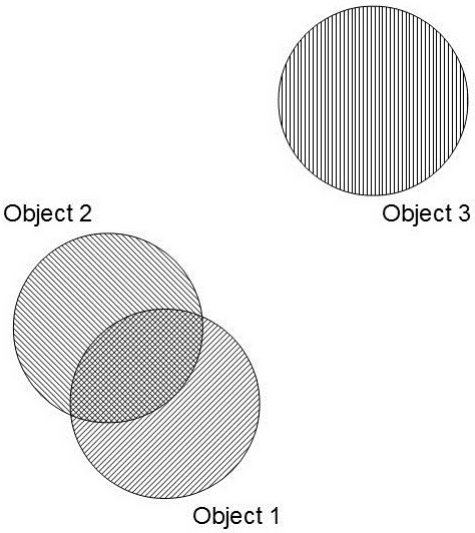


Abbildung 6.2.: Object 1 bereits vorhanden, 2 wird verworfen, 3 gespeichert.

Mehrfach wahrgenommene Objekte werden, auch unter Einfluss von Messungenauigkeiten (bis zu einem gewissen Grad), nur ein einziges Mal gespeichert.

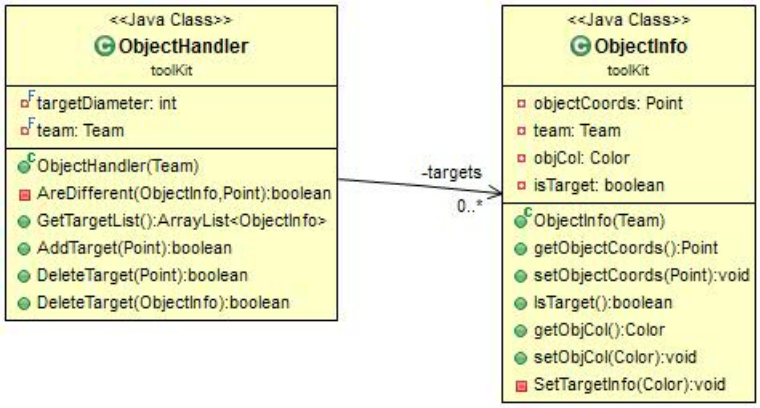


Abbildung 6.3.: Klassen-Diagramm: ObjectHandler

6.4. Coordinator

Der „Coordinator“ bekommt eine Reihe von Distanzen und Winkeln übergeben. Die Distanzen entsprechen denen, die während der Suchrotation vom wahrgenommenen Objekt zum Sensor gemessen wurden. Die Winkel beschreiben „wann“ eine bestimmte Distanz zu einem Objekt im Verlauf der Rotation gemessen wurde.

Distanzen und Winkel die ihren Nachbarn ähnlich ($|n_i - n_{i+1})| < \epsilon$) sind, beschreiben ein Objekt während sich nicht ähnelnde ($|n_i - n_{i+1}| > \epsilon$) Werte in der Reihe, als Grenzen zwischen den einzelnen Objekte dienen.

$$
\begin{array}{rl}
Distanzen(mm): & [\ 1\ 1\ 1\ 1\ 6\ 6\ 6\ 6\ 6\ 6\ 3\ 3\ 3\ ] \\
Winkel(Grad): & [\ 5\ 6\ 7\ 8\ 82\ 83\ 84\ 85\ 86\ 87\ 317\ 387\ 319\ ] \\
 & \quad \underbrace{\qquad\qquad\qquad}_{Objekt1} \quad \underbrace{\qquad\qquad\qquad\qquad\qquad}_{Objekt2} \quad \underbrace{\qquad\qquad\qquad}_{Objekt3}
\end{array}
$$

Für jedes Objekt wird mit Hilfe der aktuellen Position des Roboters, sowie Winkel und Distanz zum Objekt, ein Punkt berechnet, welcher die Position des Objektes durch Koordinaten beschreibt.

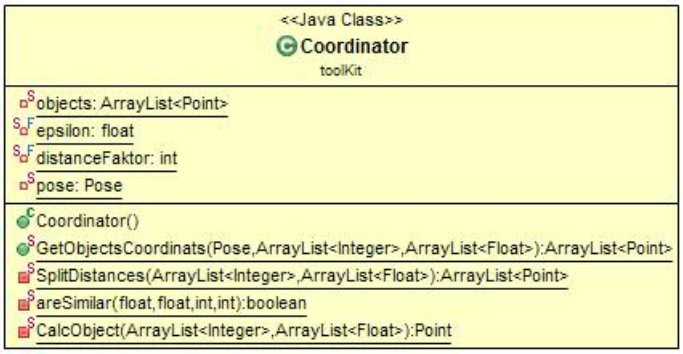


Abbildung 6.4.: Klassen-Diagramm: Coordinator

6.5. Pilot

Der „Pilot“ ist ursprünglich eine von LeJOS [6] stammende Klasse, welche dazu dient, einen Roboter mit zwei motorisierten Rädern und einem Stützrad zu bewegen. Für den Workshop wurde er aber so modifiziert, dass relativ komplexe Bewegungsabläufe möglichst einfach bewerkstelligt werden können. Des Weiteren wurde die Möglichkeit einen Listener zu registrieren, welcher die Odometry (3.4) des Roboters verfolgt, fest integriert. Der Pilot dient also dazu den Roboter zu bewegen, als auch Positionsangaben zurückzugeben.
Der Pilot existiert in zwei Ausführungen: Die Grundlegende unterstützt nur elementare Bewegungsabläufe wie „rotieren“, oder „forwärts fahren“. Es liegt also beim Benutzer, die zurückzulegende Strecke zum Ziel und die nötigen Rotationswinkel zu berechnen.
Die zweite Ausführung vereinfacht den oben beschriebenen Piloten und bietet die Möglichkeit, automatisch von der aktuellen Position aus, zu einem gegebenen Punkt P zu reisen. Ein Punkt wird dabei mit Koordinaten beschrieben: P(x,y). Die Einheit, in welcher die zurückzulegende Strecke berechnet wird, bestimmt der Benutzer über die Angaben von

Raddurchmesser und Radstand. Diese Größen sind nötige Größen zur Berechnung der Odometry (3.4).

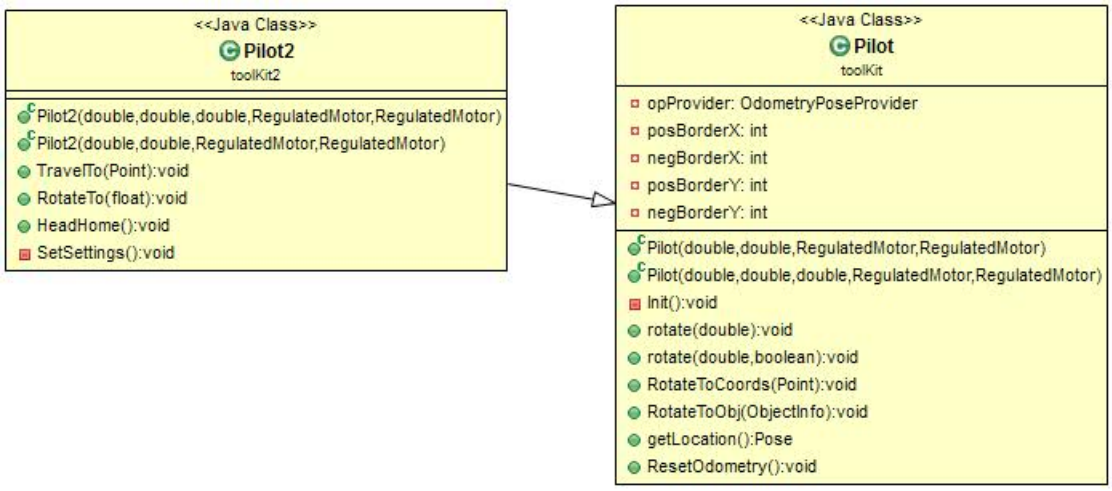


Abbildung 6.5.: Klassen-Diagramme: Pilot und Pilot2

6.6. Grappler

Auch diesen Baustein gibt es in zweifacher Ausführung: In der Basisausführung wird nur Greifen bzw. Heben und Senken bzw. Loslassen des Greifarms unterstützt. Die Zweite Ausführung vereinfacht wieder die nötigen Vorgänge zum Aufnehmen und Abstellen einer Boje. Zum Greifen muss der Roboter weit genug von der Boje entfernt sein, um den Greifarm senken und öffnen zu können. Zum Aufnehmen der Boje muss sich der Roboter zunächst ein Stück auf die Boje zubewegen, damit diese beim Schließen der Zange auch tatsächlich von dieser gegriffen wird. Zum Abstellen einer Boje ist nichts weiter zu beachten, doch um den Greifarm wieder in Ausgangsposition zu bewegen, muss sich der Roboter zunächst von der Boje entfernen, um anschließend den Greifarm anheben zu können, ohne die Boje wieder zu greifen.
In der Basisausführung des „Grapplers“ muss der Benutzer sich diese Vorgänge selbst überlegen und zusammenbauen, während die zweite Variante dies automatisch übernimmt.

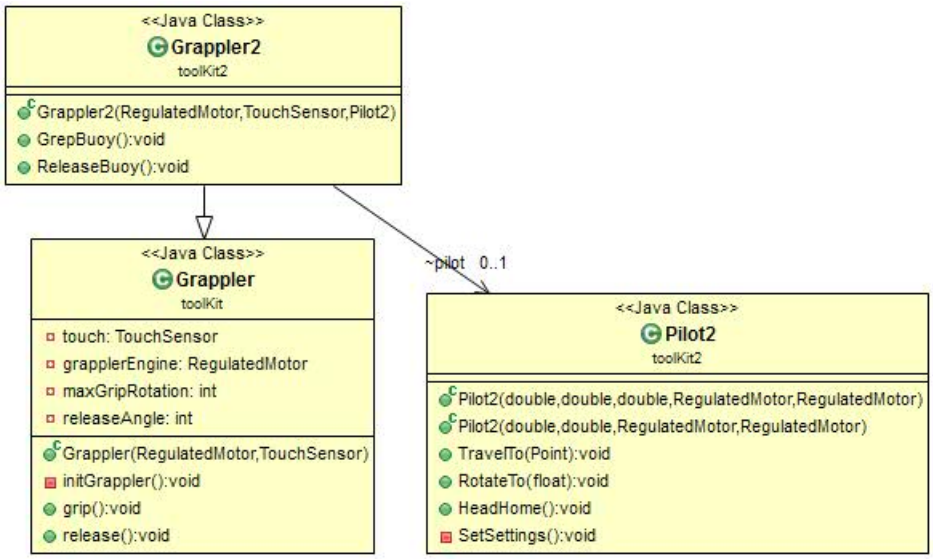


Abbildung 6.6.: Klassen-Diagramm: Grappler und Grappler2

6.7. OdometryResetListener

Der OdometryResetListener dient nur dazu, einen Tastendruck (ENTER) auf dem Roboter abzufangen und ein Signal zu geben, dass der Benutzer die Odometry (3.4) neu initialisieren möchte. Da Ungenauigkeiten in den Radabmessungen, im Radstand oder auch äußere Einflüsse wie ein rutschiger Untergrund, einen sich fortpflanzenden Fehler in der Odometry bewirken, wird diese immer weiter verfälscht. Durch eine neue Initialisierung der Odometry, wird dieser Fehler behoben bis er sich wieder neu aufgebaut hat. Zum Neuinitialisieren der Odometry, ist der Roboter von Hand in der Basis zu platzieren.

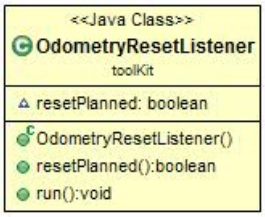


Abbildung 6.7.: Klassen-Diagramm: BorderLines

6.8. BorderLines

Dieser Baustein dient dazu, die aktuelle Position des Roboters oder der gefundenen Bojen/Objekte, gegen die gesetzten Spielfeldgrenzen zu prüfen. Bojen/Objekte außerhalb des Spielfeldes können als ungültig erklärt und Reisestrecken so verkürzt werden, dass das Ziel

noch innerhalb der Grenzen liegt.

Abbildung 6.8.: Klassen-Diagramm: BorderLines

6.9. ObjectFinder

Der „ObjectFinder“ dient dazu, mit Hilfe des Ultraschallsensors Objekte zu lokalisieren. Der „ObjectFinder“ nutzt dabei den Pilot (6.5), um den Roboter einmal um die eigene Achse zu rotieren. Während dieser Rotation wird vom Ultraschallsensor die Umgebung abgetastet. Softwareseitig werden aber, zu Gunsten des Workshops, nur Objekte in einer Distanz von maximal 50 cm zum Sensor beachtet. Diese Beschränkung soll verhindern, dass das Spielfeld, in welchem sich der oder die Roboter später bewegen sollen, nicht zu schnell erkundet werden kann. Wurde ein Objekt im innersten der Kreise (6.9) gefunden, wird eine Repräsentation durch Koordinaten (siehe Kapitel „Coordinator“) berechnet, und überprüft ob das Objekt bereits bekannt ist oder gespeichert werden soll (siehe Kapitel „ObjectHandler“).

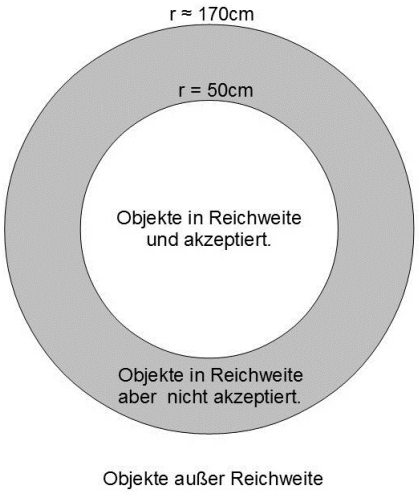


Abbildung 6.9.: Suchkreis des ObjectFinders

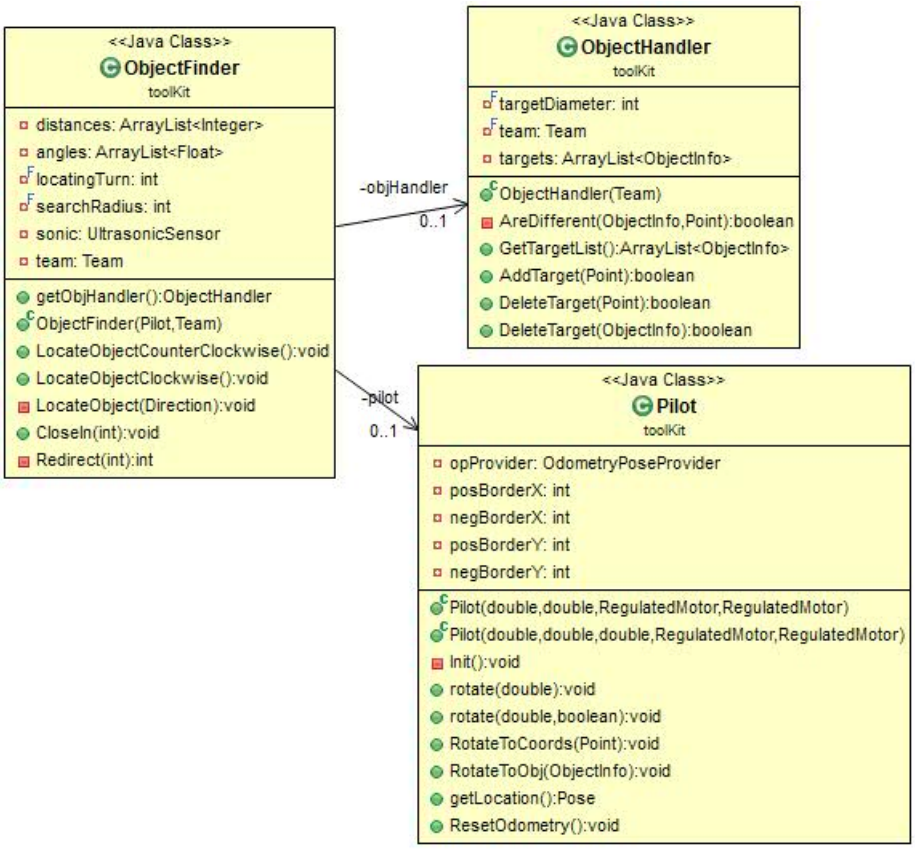


Abbildung 6.10.: Klassen-Diagramm: ObjectFinder

6.9.1. BuoyLocator

Der „BuoyLocator“ erweitert den ObjectFinder und vereinfacht die nötigen Schritte, eine Boje zu lokalisieren und sich dieser gegebenenfalls zu nähern. Der „BuoyLocator“ entscheidet dabei selbstständig, ob die gefundene Boje die Spielfeldgrenzen verletzt oder aufgenommen werden kann. Falls mehr als nur eine einzelne Boje lokalisiert wird, wird nur die erste Gültige, der gefundenen Bojen behandelt, während der Rest verworfen wird:
Gerade bei mehreren Robotern auf dem selben Spielfeld, werden Bojen oftmals zu ähnlichen Zeitpunkten lokalisiert und in ihrer Position verändert. Eine Boje B die ursprünglich an Punkt P vorzufinden war, könnte sich jetzt an Punkt Q befinden, während man selbst Boje A zur eigenen Basis bringt. Die ursprünglichen Koordinaten der Boje B wurden verändert. Um das Geschehen zu vereinfachen, wird Boje B also gar nicht erst beachtet.

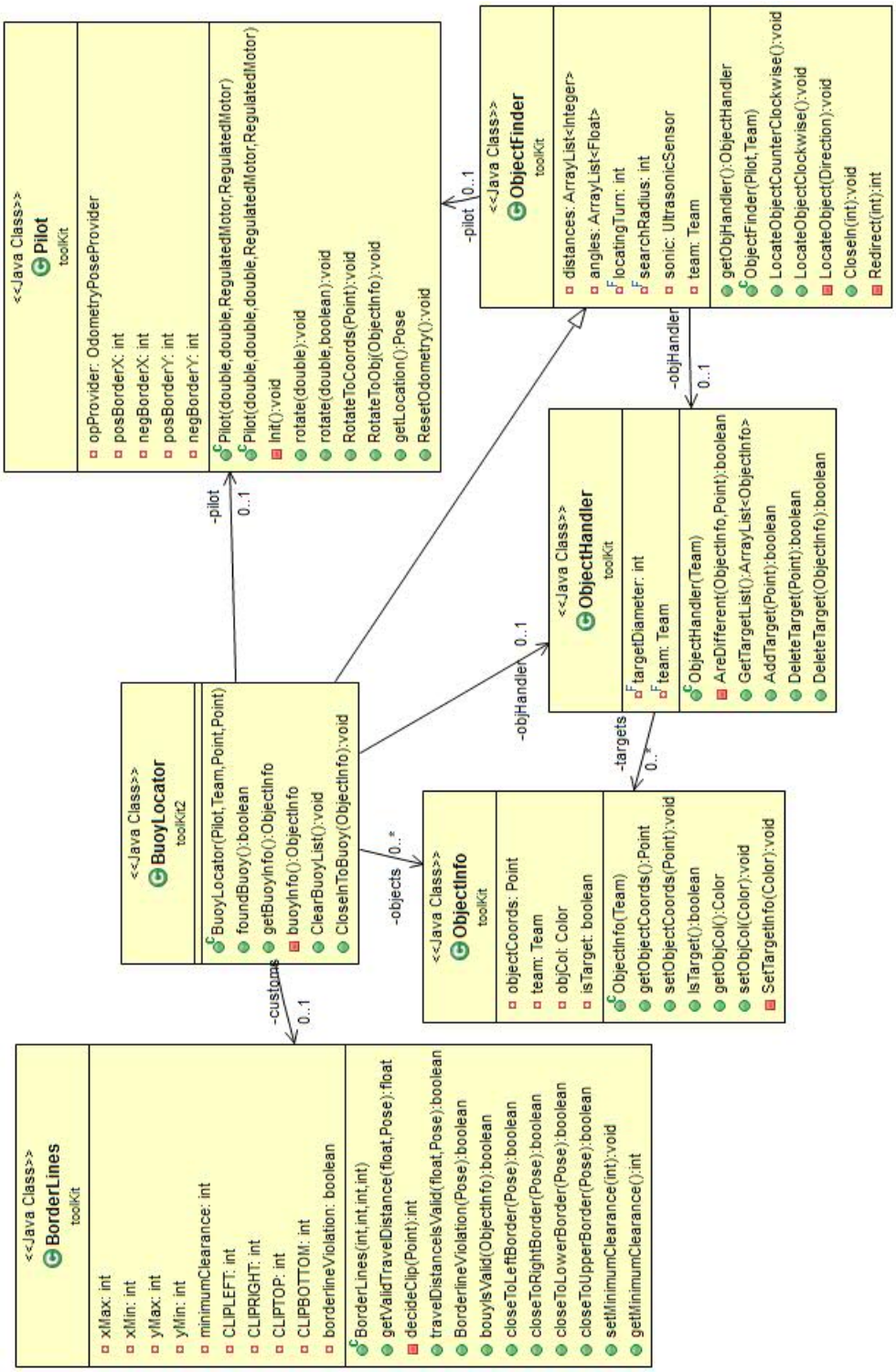


Abbildung 6.11.: Klassen-Diagramm: BuoyLocator

7. Szenarien

Dieses Kapitel beschreibt die Entwicklung der verschiedenen Szenarien, nennt die Gründe warum manche verworfen und welches Szenario am Ende in den Workshop übernommen wurde.

7.1. Verworfene Szenarien

Zu Beginn der Diplomarbeit entstanden mehrere Szenarien, von denen sich aber schlussendlich nicht alle haben umsetzen lassen:

7.1.1. Capture The Flag (CTF) und Abwandlungen

klassisches CTF

Schon in der Antike wurden Banner oder Flaggen eines Feindes gestohlen oder auf dem Schlachtfeld erobert, um die Moral des Gegners zu brechen oder schlichtweg eine Trophäe zu ergattern. Heutzutage wurde daraus ein Spiel: „Capture the Flag“ oder kurz „CTF“. Ziel des Spieles CTF ist es, die Flagge des gegnerischen Teams aus dessen Basis zu entwenden und zur eigenen zu bringen. Eine gestohlene Flagge kann dem Träger oder Dieb abgenommen und zur eigenen Basis zurück gebracht werden. Durch wiederholtes Stehlen der gegnerischen Flagge können Punkte für das eigene Team gewonnen werden, welche schlussendlich zum Sieg führen. Je nach Regelwerk und der Größe des Spielfeldes, kann ein Spiel auch schon durch die einmalige Eroberung der gegnerischen Flagge gewonnen werden.
Für den in dieser Diplomarbeit erstellten Workshop, stellt CTF ein interessantes Szenario dar, welches nach einigen Anpassungen als abschließender Wettkampf dienen sollte.

Das beschriebene Szenario stellte sich jedoch als zu komplex für die zugrundeliegende Sensorik heraus:
In verschiedenen Probeläufen reagierten die Systeme zu langsam oder zu ungenau, um eine Flagge schnell und zuversichtlich von einem Roboter zu unterscheiden. Einen „Flaggenträger“, den Dieb, als solchen von den eigenen und gegnerischen Einheiten schnell und auf Distanz zu erkennen, ist mit den zur Verfügung stehenden Sensoren fast nicht möglich. Die gestohlene Flagge dem Dieb wieder abzunehmen, stellt zusätzliche Ansprüche an die Bewegungsfreiheit des Greifarms und lässt sich mit nur einem Servomotor (die restlichen zwei sind für die Fortbewegung des Roboters verantwortlich) nicht umsetzen.

Ein Ansatz diese Problem zu lösen war es, mit Hilfe von RF-ID einen einfachen Datenaustausch zwischen zwei Robotern aufzubauen. Über diese Datenverbindung sollte ein Signal an den Flaggendieb abgesetzt werden, die gestohlene Flagge fallen zu lassen. Aber auch dieser Ansatz ist zu sehr von der genauen Positionsbestimmung eines Roboters und der sehr begrenzten Reichweite der RF-ID-Chips abhängig.
Die von CTF gestellten Anforderungen an Präzision, Reaktionszeit und Geschwindigkeit, sind von den zur Verfügungen stehen Sensoren und Aktoren leider nicht zu erfüllen.

Abgewandeltes CTF

Um das obige Szenario etwas zu vereinfachen, sollte eine einzelne, an beliebiger Position auf dem Spielfeld abgestellte, Flagge gesucht und geborgen werden. Ist die Flagge gefunden und zur eigenen Basis zurück gebracht worden, erhält das jeweilige Team einen Punkt und die Flagge muss von Hand wieder an eine beliebige Position des Spielfeldes gesetzt werden. Hat eine Einheit die Flagge gefunden, kann eine Einheit des gegnerischen Teams versuchen, diese dem ursprünglichen Träger wieder abzunehmen.
Die Schwierigkeit, die eigene Flagge zu schützen während gleichzeitig nach der gegnerischen gesucht werden muss, fällt bei dieser Abwandlung weg. Noch immer bestünde aber die oben genannte Notwendigkeit, Flaggen und Roboter sehr genau in ihrer Position zu bestimmen und Flaggenträger von normalen Einheiten zuverlässig zu unterscheiden. Auch die notwendige Bewegungsfreiheit des Greifarmes kann nach wie vor nicht umgesetzt werden.

7.2. Umsetzbare Szenarien

7.2.1. Hindernisparcours

Dieses Szenario sollte von den jeweiligen Teilnehmern möglichst schnell und auf Zeit absolviert werden. Verschiedene Hindernisse wie Unebenheiten, auszuweichenden Objekten oder Steigungen, sollen dabei den Roboter des jeweiligen Teams den Weg von Start- zu Ziellinie erschweren. Je schneller und zuverlässiger ein Roboter durch den Parcours kommt, desto besser dessen Platzierung am Ende.
Bei diesem Szenario müssen Flaggen nicht von weiteren Robotern unterschieden werden und generell sind die Anforderungen von Reaktionszeit und Genauigkeit an die Sensorik, bei weitem nicht so hoch wie bei den oben dargestellten Szenarien. Mögliche Probleme lägen bei der zuverlässigen Objekterkennung, je nach Hindernisgeometry oder Oberflächenbeschaffenheit.

Der Hindernissparcours ist zwar prinzipiell umsetzbar, dafür aber nicht sonderlich unterhaltsam. Hinzu kommt die Möglichkeit, die vom Roboter zu wählende Route durch den Parcours voraus zu berechnen, was den Sinn des Szenarios etwas in Frage stellt.

Mehr zu dieser Thematik ist im Kapitel „Probleme, Schwierigkeiten und Herausforderungen“ zu finden.

7.2.2. Bojenjagd

Die „Bojenjagd“ entstand, als alle als zu komplex geltenden Aspekte des CTF oder des abgewandelten CTF entfernt und die gut umsetzbaren Aspekte hervorgehoben wurden:
Statt nur einer Flagge, sind mehrere Flaggen auf dem Spielfeld verteilt. Eine Flagge ist zwischen einem und mehreren Punkten, in Abhängigkeit der Flaggenfarbe, wert. Punkte gibt es aber nur, wenn eine Flagge erfolgreich gegriffen und zur eigenen Basis gebracht wurde. Eine vom Roboter momentan angehobene Flagge kann und darf nicht von einer gegnerischen Einheit gestohlen werden. Ziel ist es, vor Ablauf der Zeit möglichst viele Flaggen und Punkte zu sammeln.
Bei diesem Szenario muss der Dieb der eigenen Flagge nicht aufgehalten oder erst als solcher identifiziert werden. Der Greifarm muss nur heben, senken und greifen können und auch die Anforderungen von Reaktionszeit und Genauigkeit an die Sensorik, sind nicht zu hoch. Flaggen wurden durch „Bojen“ ersetzt, da diese Umschreibung auf das tatsächliche, auf dem Spielfeld stehenden, Objekt eher zutrifft.
Die Roboter der Teams müssen nicht unbedingt alle zur gleichen Zeit antreten. Eins gegen Eins in im K.O.- oder Turnier-Verfahren ist sinnvoller, da weniger Roboter auf dem Spielfeld auch weniger Störsignale für die Ultraschallsensoren bedeuten.

Abwandlung der Bojenjagd

Das Vorliegende Szenario kann ohne Weiteres so verändert werden, dass die Teilnehmer nacheinander und auf Zeit antreten. Ziel ist es in möglichst kurzer Zeit so viele Bojen und Punkte wie möglich zu sammeln. Die Platzierung verläuft nach gesammelten Punkten und Zeitdauer.
Diese Abwandlung der Bojenjagd vereinfacht das ursprüngliche Szenario: Immer nur eine einzelne Einheit ist auf dem Spielfeld und auf Bojenjagd. Dadurch können sich Ultraschallsensoren untereinander nicht stören (siehe auch Kapitel „Verhalten von Ultraschallwellen“) und Bojen müssen nicht von Robotern unterschieden werden.
Des Weiteren können, je nach gewünschtem Schwierigkeitsgrad oder zur Verfügung stehender Zeit, die Farben der Bojen mit unterschiedlich vielen Punkten belegt oder eben missachtet werden. Ein von der Bojenfarbe abhängiger Punktewert, könnte das Spielgeschehen sehr beeinflussen. Einige Teams könnten planen zunächst nach den wertvollen Bojen zu suchen und alle anderen vorerst zu ignorieren, während andere Teams jede Bojen greifen, auf die sie stoßen.
Die auf dem Spielfeld verteilten Bojen können auch, je nach Farbe, einem Team zugeordnet werden. So würde „Team-Blau“ nur für blaue und „Team-Grün“ nur für grüne Bojen Punkte bekommen können. Jedoch könnte Team-Blau grüne Bojen jagen und am Spielfeldrand „verstecken“ oder die Boje einfach um schmeißen, damit der Roboter des Gegnerteams diese nicht mehr oder nur schwer greifen kann.

Anmerkungen

Je nach Suchradius des Roboters, ist die Spielfeldgröße des Szenarios zu beachten. Der Suchradius ist standardmäßig auf 50 cm gesetzt. Diese Einstellung soll von den Workshopteilnehmern nicht geändert werden, die Möglichkeit dazu besteht jedoch in der Klasse „toolKit.ObjectFinder“. Ein Spielfeld von zwei auf zwei Metern ist bei Standarteinstellungen relativ klein, aber dennoch akzeptabel, da Bojen gerade an den Suchkreisrändern teilweise gar nicht oder nur ungenau erkannt werden. Die Teilnehmer werden daher oftmals zu überlappenden Suchkreisen neigen, falls sie auf diese Problematik hingewiesen wurden oder in der Analyse erkannt haben. Das genannte Spielfeld sollte in diesem Fall also groß genug sein.

Regelwerk der Bojenjagd

- Die Spielfeldgrenzen dürfen von einem Roboter nicht verletzt werden.
- Bojen außerhalb der Spielfeldgrenzen, gelten als ungültig. Sollte ein Roboter eine solche Boje aufgreifen, befindet er sich außerhalb des Spielfeldes und verstößt gegen obige Regel. Für die ungültige Boje gibt es jedoch keinen Punktabzug.
- Bojen dürfen und können aus der gegnerischen Basis gestohlen werden (nur relevant falls ein Gegner auf dem Spielfeld vorhanden ist)
- Sollte eine in der Basis befindliche Boje, unabsichtlich aus dem Spielfeld oder der Basis geschoben werden, zählt diese Boje weiterhin als Punkt für das eigene Team. Der Unparteiische hat hierbei das letzte Wort.
- Zwei ineinander verhakte Roboter, sind vom Unparteiischen zu trennen und in der jeweiligen Basis zu platzieren (Odometry-Reset beachten).

8. Probleme, Schwierigkeiten und Herausforderungen

> Probleme kann man niemals mit derselben Denkweise lösen, durch die sie entstanden sind.
>
> *(Albert Einstein)*

Dieses Kapitel beschreibt die Probleme und Schwierigkeiten die es zu lösen, aber auch zu akzeptieren galt und stellt die Herausforderungen dar, welche hard- und softwareseitig gemeistert werden mussten.

8.1. Entwurf und Bau des Workshop-Roboters

Der im Workshop zum Einsatz kommende Roboter, muss einige Anforderungen erfüllen:

- Rotation (Kurvenfahrt mit Radius = 0) in beide Richtungen für einfache und exakte Ausrichtung auf Bojen
- Vorwärts und rückwärts fahren um Distanz zu Bojen genau variieren zu können.
- Erkennung der Umgebung / von Hindernissen um Bojen zu lokalisieren. Muss auf ausreichend große Distanzen möglich sein.
- Heben, senken und greifen einer Boje durch Greifarm mit nur einem Motor

Da zur Fortbewegung und Rotation des Roboters schon allein zwei, der maximal drei zur Verfügung stehenden Servomotoren benötigt werden, war ein Greifarm zu entwickeln, welcher mit nur einem einzigen Antrieb heben, senken und greifen kann (siehe Kapitel „Greifarm“).
Distanzen von mehr als 2 - 4 cm können nur vom Ultraschallsensor abgetastet werden. Dieser muss dabei so angebracht werden, dass er am Boden stehende Objekte erkennt, ohne dabei von angehobenen Bojen verdeckt zu werden. Um Bojen-Farben unterscheiden zu können, muss ein Farbsensor am Roboter angebracht sein. Dieser soll möglichst angehobene, als auch am Boden befindliche Bojen abtasten können. Die Positionierung des Sensors muss also gewährleisten, dass Bojen unabhängig von ihrer Position um den Roboter abgetastet werden können (so lange sie in Reichweite sind). Eine Montierung an der Greifzange bietet sich hier an: Gegriffene Bojen befinden sich direkt vor dem Sensor und am Boden befindliche Bojen können vom Sensor erfasst werden, wenn dieser vom Greifarm entsprechend in Position gebracht wird.

8.1.1. Greifarm

Der Greifarm stellte die eigentliche Herausforderung am Entwurf und Bau des Roboters dar. Wie oben schon erwähnt, werden bereits zwei von drei Servomotoren zur Fortbewegung benötigt. Der Letzte, für den Greifarm übrigbleibende Servomotor, muss den Greifarm heben, senken, öffnen und schließen. Umzusetzen ist dies mit den im Baukasten zur Verfügung stehenden Mitteln nur, wenn man den Widerstand der Greifzange nutzt, welcher Auftritt, wenn diese komplett geschlossen ist oder ein Objekt mit genügend hohem Druck gegriffen hat. Statt der Zangenbewegung wird eine Hub- oder Senkbewegung ausgeführt. Die genaue Beschreibung des Greifers ist unter Kapitel „Der Greifarm“ nachzulesen.

8.2. Grenzen der Sensorik

Die von Lego im Baukasten mitgelieferte Sensorik arbeitet leider nicht so genau wie erhofft. So sollte der mitgelieferte Helligkeitssensor beispielsweise die Farben Rot und Blau anhand deren Helligkeit unterscheiden können. Tatsächlich aber funktioniert dies nur äußerst unzuverlässig. Auch die Wahrnehmung der mitgelieferten Hindernisse (zwei Bälle, etwa der Größe eines Tischtennisballes), klappt mit Hilfe des Ultraschallsensors nur in den seltensten Fällen.
Im Folgenden werden die für den Workshop benötigten Sensoren sowie deren Problematik genauer dargestellt:

Helligkeitssensor / Farbsensor

Der Helligkeitssensor eignet sich gut um Kontraste von einander zu unterscheiden. So wird Schwarz von Weiß bei ausreichender Beleuchtung zuverlässig von einander unterschieden. Auch Farben lassen sich bei ausreichender Beleuchtung beschränkt, anhand ihres Helligkeitswertes, von einander unterscheiden. Je ähnlicher die voneinander zu unterscheidenden Farben jedoch sind, desto ähnlicher deren Helligkeitswert und desto unzuverlässiger die Unterscheidung. Ein kräftiges Rot lässt sich von einem kräftigen Blau nicht immer präzise unterscheiden. Generell werden gute Lichtverhältnisse benötigt.

- Maximale Distanz: ca. 4 bis 5 cm
- Minimale Distanz: ca. 1 cm (genügend Abstand zum Objekt, um eine ausreichende Beleuchtung zu gewährleisten)
- Unterscheidung von ausreichend starken Kontrasten
- Probleme bei Farben, deren Helligkeitswerte zu ähnlich sind
- Beispielhafte Anwendung: Linien-Folger

Da der Helligkeitssensor Farben zu unpräzise oder gar nicht unterscheidet, kommt im Workshop stattdessen ein nachbestellter Farbsensor zum Einsatz. Dieser erkennt Farben bei ausreichenden Lichtverhältnissen und Beleuchtung relativ genau, unterliegt aber sonst, bis auf diese Erweiterung, den selben Beschränkungen wie der Helligkeitssensor.

Berührungssensor

Der Berührungssensor ist relativ unempfindlich gegenüber äußeren Einflüssen. Der Sensor reagiert sobald der Taster ausreichend stark gedrückt wird. Das Signal beschränkt sich dabei auf 1 oder 0 (binär) für „gedrückt“ oder „nicht gedrückt“, eine Erfassung des anliegenden Druckes ist nicht möglich.
Der nötige Druck, um das am Taster anliegende Signal zu ändern, ist relativ hoch. Eine Boje könnte einfach umgefahren werden, ohne das der Taster reagiert. Daher kann die Positionierung des Roboter, um eine Boje aufzunehmen, nicht über den Berührungssensor verwirklicht werden.

Ultraschallsensor

Als einziger, dem Baukasten beiliegender, ist der Ultraschallsensor dazu in der Lage, Objekte auch auf Distanzen von mehr als 5 cm zu erkennen. Erkannt wird die Distanz zu einem Objekt mit Hilfe einer vom Sender ausgesandten Ultraschallwelle. Diese Welle wird vom Objekt reflektiert und vom Empfänger des Sensors aufgefangen. Anhand der gemessenen Zeit, die die Welle vom Sender bis zum Objekt und von diesem zurück benötigt, kann die Distanz zwischen Sensor und Objekt berechnet und zurückgegeben werden. Je nach Geometrie, Oberflächenbeschaffenheit und Distanz zum Objekt, funktioniert die Erfassung besser oder schlechter. Der mitgelieferte Sensor hat eine Reichweite von ca. 170 cm und gibt „255“ im Fehlerfall, oder falls nichts erkannt wird, an. Der Sensor arbeitet je nach Distanz unterschiedlich genau und verwendet Zentimeter (cm) als Einheit. Im Bereich von 0 bis 20 cm gibt der Sensor leider häufig recht ungenaue Rückgabewerte an und erkennt auch Änderungen in der Distanz oftmals nur in 5 cm Sprüngen. Von 20 cm an aufwärts, arbeitet der Sensor relativ genau, solange ein Objekt erkannt wird. Ein Objekt wird auf große Distanzen (> 50 cm) aber nicht immer erkannt. Hier spielt die Größe des Objektes eine wichtige Rolle: Je größer ein Objekt, desto zuverlässiger dessen Erkennung auf große Distanzen.
Der Ultraschallsensor arbeitet mit einem Kegel von ca. 60° vor dem Sensor, in welchem sich das Ultraschallfeld ausbreitet. Die genauerer Problematik und Eigenheiten der Ultraschalltechnik, werden in Kapitel „Verhalten von Ultraschallwellen“ beschrieben.

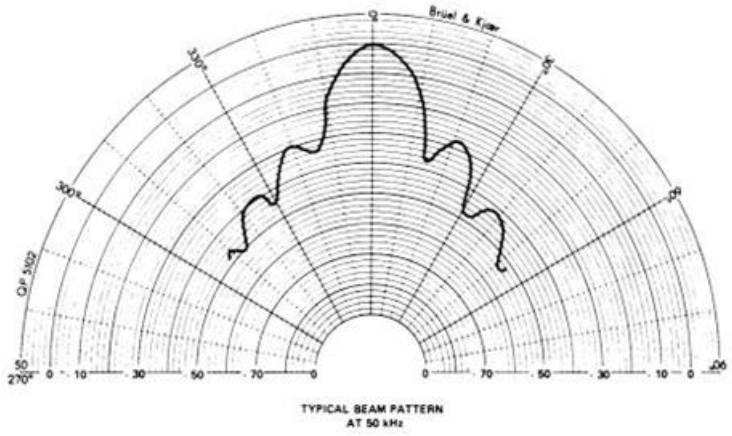


Abbildung 8.1.: Ausbreitung eines Ultraschallfeldes[2]

8.2.1. Verhalten von Ultraschallwellen

Wie oben schon kurz erwähnt, breitet sich Ultraschall, innerhalb eines Öffnungsbereiches von ca. 60° vor dem Sensor aus. Innerhalb dieses Feldes können Objekte erkannt, in ihrer Position aber nicht ohne Weiteres genauer bestimmt werden.

Ultraschallwellen legen, je nach Oberflächenbeschaffenheit oder Geometrie eines Objektes, ein unpraktisches Verhalten an den Tag. So gibt es Oberflächen die Ultraschallwellen besser oder schlechter reflektieren, oder diese gar komplett absorbieren. In solchen Fällen wird ein Objekt innerhalb des Erkennungsbereiches gut, schlecht oder unzuverlässig erkannt, unter Umständen auch gar nicht.
Bei einem sehr flachen Winkel von Objekt-Oberfläche zu Sensor, kann das Feld so reflektiert werden, dass die einzelnen Wellen nicht mehr zum Empfänger zurück gelangen. Auch hier wird das Objekt nur unzuverlässig oder gar nicht erkannt. Ein weiterer Problemfall entsteht bei einer ungünstigen Oberflächen-Geometrie, welche das Feld zerstreut, anstatt es zum Empfänger zurück zu reflektieren. Eine zuverlässige Erkennung des Objektes ist auch hier wieder nicht möglich.
Ein vierter Fall, in welchem Objekte nicht zuverlässig erkannt werden, tritt ein, wenn das Ultraschallfeld nicht direkt zum Empfänger zurück reflektiert wird, sondern über Umwege zu diesem zurück gelangt. In einem solchen Fall misst der Sensor aufgrund der mehrfach reflektierten Welle und der damit einher gehenden größeren Zeitdauer, eine größere als die tatsächliche Distanz zum Objekt.
Die folgenden drei Abbildungen zeigen Fälle, in denen Ultraschallwellen ungünstig reflektiert werden:

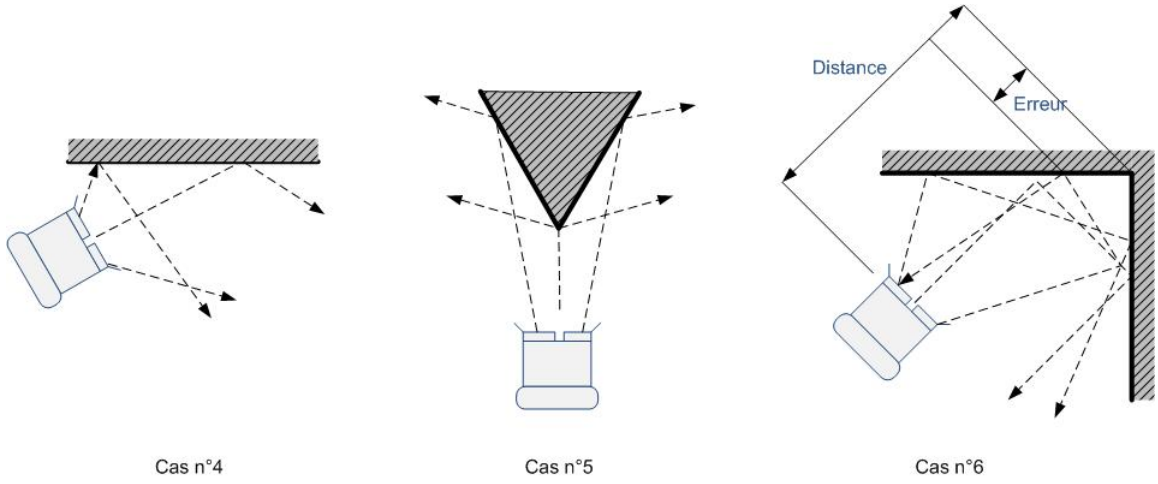


Abbildung 8.2.: Ungünstig reflektierte Ultraschallwellen[2]

Ein letztes, wenn auch Mindstorms spezifisches Problem tritt auf, wenn mehrere Ultraschallsensoren in näherer Umgebung aktiv sind. Arbeiten die Sensoren auf der selben Frequenz, so stören sie sich gegenseitig und empfangen möglicherweise Signale des jeweils anderen Sensors. Der von Lego ausgelieferte Sensor bietet weder hard- noch softwareseitige Möglichkeiten, die Frequenzen der Sensoren zu ändern. Diese Problematik ist nur durch entsprechende Gestaltung der Umgebung oder des Szenarios zu kompensieren, oder muss einfach akzeptiert werden.

8.3. Gestaltung des Workshops

Der Workshop, oder dessen Anforderungen, sind gerade für Laien doch relativ hoch. So sind Kenntnisse im Umgang mit Sprachelementen von Java oder dessen Syntax sowie der eingesetzten IDE (Eclipse) nur der Anfang. Die Teilnehmer müssen Vorgehensstrategien planen, Algorithmen entwerfen und beides schließlich implementieren. Generell steht eher zu wenig, als zu viel Zeit zur Verfügung.

8.3.1. Zeitintensiver Aufbau des Roboters

Alle Testkandidaten, welche den Roboter selbst zusammenbauten, benötigten für dessen Komplettierung ungefähr eine Stunde oder gar mehr, falls der Teilnehmer allein arbeitete. Da für den Workshop aber eher zu wenig als zu viel Zeit zur Verfügung steht, wird der Aufbau bei den meisten Workshops wohl komplett aus dem Programm gestrichen werden müssen. Alternativ könnte man die Teilnehmer nur einen Teil, beispielsweise den Greifarm, des Roboters bauen lassen, während ein Grundmodell gestellt werden würde.
Der Zusammenbau des Roboters stellt, den Pilotworkshops zu folge, aber einen sehr motivierenden Part des gesamten Programms dar, so fragten alle Teilnehmer, ob sie den Roboter

selber bauen dürften. Aus motivationstechnischen Gründen wäre ein Kompromiss zwischen Zeitaufwand und Aufbau interessant.

8.3.2. Die Wahl der Bojen

Wie in „Verhalten von Ultraschallwellen“ beschrieben, ist die Oberflächenbeschaffenheit eines Objekts ein wichtiges Detail bei der Wahl der Boje. Die Boje muss aus möglichst jeder Position auf dem Spielfeld lokalisiert werden können und sollte demnach keine scharfen Ecken in Verbindung mit angrenzenden flachen Oberflächen haben. Eine viereckige Bojenform stellte sich als problematisch heraus, während Dreiecke nur schwer vom Sensor erkannt werden konnten (siehe auch Kapitel „Verhalten von Ultraschallwellen“). Die den Baukästen beiliegenden farbigen Bälle waren ebenfalls ungeeignet, da sie aufgrund ihrer Geometrie das Ultraschallfeld in alle Richtungen zerstreuen.
Die schließlich gewählte Boje wird unter Kapitel „Die Bojen“ genauer Beschrieben.

8.3.3. Kick-Off-Präsentation

Der, in der am Anfang des Workshops stehenden Präsentation (5.3), behandelte Stoff ist doch sehr Umfangreich für die vorgesehene Zeit. Die Pilotworkshops (5.6) zeigten auf, dass nur Teilnehmer mit entsprechendem Hintergrundwissen, der Präsentation ohne Probleme folgen und das vermittelte Wissen umsetzen konnten. Da aber Laien ebenfalls zur Zielgruppe gehören, ist die Präsentation so zu gestalten und mit didaktischen Stilmitteln zu bestücken, dass nach der Präsentation alle Teilnehmer die möglichst gleichen Chancen haben.
Für den Workshop nötiges Wissen:

- Java-Syntax
- Schleifen
- if-/else-Konstrukte
- Verknüpfen von Bedingungen durch Logik-Operatoren
- Definition von Variablen
- Instanziierung von Objekten
- Definition und Aufruf von Methoden
- (Umgang mit Eclipse (IDE))

Diese Dinge können durch die besagte Präsentation abgedeckt werden. Passende Beispiele sollen dabei helfen.

9. Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde mit Hilfe von LEGO-Mindstorms ein Workshop verwirklicht, welcher die Aufmerksamkeit von Studieninteressenten auf die Softwaretechnik lenken soll. Da sich der Workshop um die Softwaretechnik dreht, vermittelt er den Teilnehmern einige wenige Dinge, die im Software-Engineering üblich sind und verpackt diese möglichst interessant.
Der erste Schritt war es, einen Roboter zu entwickeln, welcher im Workshop zum Einsatz kommen soll. Dieser Roboter soll in besagtem Workshop von den Teilnehmern mit Hilfe von Java programmiert werden. Um den Teilnehmern ihre Arbeit zu erleichtern, wurden speziell für den Workshop, einige Softwarebausteine entwickelt. Diese geben dem Benutzer, in diesem Fall der Teilnehmer, die Möglichkeit, die eigentlich sehr komplexe Robotersteuerung, auch ohne Fachwissen zu verwirklichen. Das dazu nötige Wissen wird zu Beginn des Workshops vermittelt und mit Hilfe von Beispielen, Vorlagen und Handouts verdeutlicht.
Am Ende des Workshop, treten die Roboter der einzelnen Teams in einem eigens konzipierten Szenario gegeneinander an, welches das große Finale des Workshops darstellt.

Ausblick

Der Workshop könnte im Lauf der Zeit weiter ausgebaut werden: Mehr Softwarebausteine für mehr Möglichkeiten in der Gestaltung der Robotersteuerung, weitere Szenarien wären ebenfalls denkbar. So könnte man sich auf lange Sicht für jede Teilnehmergruppe, einen angemessen anspruchsvollen Workshop anfertigen. Letzterer könnte dann nicht nur für Studieninteressenten, sondern möglicherweise auch im Rahmen der ein oder anderen Vorlesung zum Einsatz kommen. Außerdem wird LEGO-Mindstorms ständig weiterentwickelt und würde dem Workshop in Zukunft vielleicht völlig neue Möglichkeiten bieten. Die Motoren werden Leistungsfähiger, die Sensoren genauer und vielfältiger. Eine Kamera in Verbindung mit genügend Speicherplatz, würde einen weiteren Weg zu Erkennung der Umgebung ermöglichen. Auch ein Lasersensor wäre sehr interessant, da äußerst präzise. Eine Kombination aus Ultraschall- und Laser-Sensor könnte Beispielsweise ein weites Feld, schnell nach Objekten abtasten und die genaue Position und Entfernung zum Objekt mit Hilfe des Lasersensors genau bestimmen. Möglichkeiten zur Weiterentwicklung des Workshops würden in Zukunft also zu genüge bestehen.

A. Anhang

A.1. Protokolle der Pilotworkshops

Protokoll - Pilotworkshop 02.01.2012

Raimund Metzler

Teilnehmer:

- Teilnehmer 1
  - Informatikstudium
  - Erfahrung mit Eclipse
  - Begrenzte Java-Kenntnisse

Gesamtdauer des Workshops: 13:40 - 18:00 Uhr (inkl. Feedback und konstruktiver Kritik)

Präsentation

Dauer: 13:45 - 14:05 Uhr
Bemerkung:
Der Teilnehmer hatte keine Schwierigkeiten dem Inhalt der Präsentation zu folgen. Unklarheiten wurden mit kurzen Fragen geklärt.

Feedback

- Seitenzahlen für bessere Übersicht
- Zu viel Text auf den Folien (Stichwörter)
- Code-Beispiele als solchen kennzeichnen und besser hervorheben.
- Mehr Beispiele für die einzelnen Themen der Präsentation (Die Teilnehmer sind Laien, keine Informatikstudenten)
- Verwendete Fachwörter erklären
- Bessere Unterscheidung von Methoden mit und ohne Rückgabewert
- Logik-Operatoren behandeln (&&, ||, ==, !)

- Darstellung von Koordinaten-System mit Dreh-Winkeln des Roboters ist missverständlich In zwei Zeichnungen aufspalten.
- Farben in der Abbildung des Suchkreises tauschen

Zusammenbau des Roboters

Dauer: 14:15 - 15:35 Uhr
Bemerkung:
Der Teilnehmer hatte teilweise Schwierigkeiten dem Bauplan zu folgen, wenn Bausteine rotiert oder die Perspektive der Abbildungen geändert wurde. Die einzelnen Schritte konnten sonst gut nachvollzogen werden.

Feedback

- Schritte / Abbildungen durchnummerieren
- Rotationen mit Pfeilen zwischen den Abbildungen andeuten.

Entwurf, Planung und Implementierung

Dauer: 15:45 - 18:00 Uhr
Bemerkung:
Teilnehmer ging mehr nach „Code & Fix“ vor, als zu planen. Entwurf / Planung wurde nur auf Aufforderung des Workshop-Leiters (kurz) vollzogen.

Feedback

Der Teilnehmer hatte Spaß am Workshop, fand diesen aber doch relativ anspruchsvoll für die vorgesehene Zeit. Laien würde möglicherweise überfordert sein.

Ergebnis

Der Teilnehmer konnte alle Phasen des Workshops erfolgreich absolvieren und auch das Szenario am Ende bewältigen.

Protokoll - Pilotworkshop 06.01.2012

Raimund Metzler

Teilnehmer:

- Teilnehmer 1
  - kein Hintergrundwissen vorhanden
- Teilnehmer 2
  - kein Hintergrundwissen vorhanden
- Teilnehmer 3
  - Grundlegende Kenntnisse in Java.

Gesamtdauer des Workshops: ...

Präsentation

Dauer: 17:30 - 18:00 Uhr
Bemerkung:
Der Inhalt der Präsentation war für die beiden Teilnehmer ohne Vorwissen zu viel. Beide hatten Schwierigkeiten den Stoff zu verarbeiten. Der dritte Teilnehmer hatte, aufgrund seines Vorwissens, keinerlei Probleme der Präsentation zu folgen.

Feedback

- Beispiele auf den Workshop / Roboter beziehen
- „Live-Beispiele“ als Hilfe um das Wissen zu verfestigen und einen praktischen Bezug zu erstellen.

Zusammenbau des Roboters

Dauer: 18:00 -18:50 Uhr
Bemerkung:
Die drei Teilnehmer teilten sich in Untergruppen auf und konnten teile des Roboters parallel zusammenbauen.
Es traten kaum Schwierigkeiten auf, eine Seitenzahl auf jeder Folie hätte aber für mehr Übersicht gesorgt.

Feedback

- Alle Schritte durchnummerieren.
- Seitenzahl auf jeder Folie
- Abbildung des fertigen Roboters am Anfang des Bauplan (Ziel vor Augen)

Entwurf, Planung und Implementierung

Dauer: 18:55 - 19:15 Uhr
Bemerkung:
Der Sinn dieses Schrittes war anfänglich nicht allen Teilnehmern klar. Auch das weitere Vorgehen war etwas unklar (vermutlich wegen mangelnder Erfahrung?).

Schwierigkeiten traten von Anfang auf:

- Definition von Variablen: unklar
- Instanziierung von Objekten: unklar
- Umgang mit der IDE anfänglich problematisch

Die zunächst geplante Strategie zur Erkundung des Spielfeldes, wurde im Laufe der Implementierung verworfen und völlig anders umgesetzt.

Feedback

Zwei der drei Teilnehmer meinte, überfordert gewesen zu sein. Und das an jedem Punkt dieser Phase. Der dritte Teilnehmer konnte Fortschritte auch ohne Hilfe verzeichnen, fand die Implementierung aber doch sehr anspruchsvoll.

Ergebnis

Die fertige Robotersteuerung war nicht wirklich funktionstüchtig und brachte nur zufällig die Eine oder Andere Boje erfolgreich in die Basis.
Bemerkungen:
Der Workshop sollte noch weiter vereinfacht werden.
Ein Zeitrahmen von drei Stunden ist mit Laien vermutlich nicht einzuhalten.

A.2. Vorlagen und Beispiele

A.2.1. Mögliches Beispiel der Präsentation

```java
/**
 * Defines the package, this class is part of.
 * NOT IMPORTANT FOR THE WORKSOP
 */
package main;

/**
 * Import necessary libraries.
 * NOT IMPORTANT FOR THE WORKSHOP
 */
import java.util.ArrayList;

import lejos.geom.Point;
import lejos.nxt.Button;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.TouchSensor;
import lejos.nxt.UltrasonicSensor;
import lejos.robotics.navigation.Pose;
import toolKit.Enums.Team;
import toolKit.ObjectInfo;
import toolKit.OdometryResetListener;
import toolKit2.BuoyLocator;
import toolKit2.Grappler2;
import toolKit2.Pilot2;

/**
 * This class represents the robot-control created as example during the Kick-Off-Presentation.
 * Every step the robot takes is defined inside this class.
 * NOTE: This is just an example and does not solve the scenario. You can use it to create
 * your own robot-control or just improve this one.
 */
public class WS_fertigesBsp {
    /**
     * Main class, called and executed be the lejos-vm on your robot.
     */
    public static void main(String[] args) {
        /**
         * Variable used as parameter for the pilot, instantiated later on.
         * Wheel-diameter = 56mm.
         */
        double wheelDiameter = 56.0;

        /**
         * Variable used as parameter for the pilot, instantiated later on.
         * Wheel-base = 120mm
         */
        double wheelBase = 120.0;

        /**
         * The pilot moves and rotates the robot and keeps track of its whereabouts.
         * Parameter:
         * wheelDiameter: The Diameter of the robot's wheels in mm.
         * wheelBase: The robot's wheelbase in mm.
         * leftMotor: The motor responsible for the left wheel (Motor.A)
         * rightMotor: The motor responsible for the right wheel (Motor.B)
         */
        Pilot2 pilot = new Pilot2(wheelDiameter, wheelBase, Motor.A, Motor.B);

        /**
         * The touchsensor is part of the grappler (instantiated later on) and shall
```

```
62          * be connected to sensorport 4 (SensorPort.S4)
63          * Parameter:
64          * port: The sensortport, the sensor is connected to (SensorPort.S4)
65          */
66         TouchSensor touch = new TouchSensor(SensorPort.S4);
67
68         /**
69          * The grappler can grab an object like a buoy and release it afterwards.
70          * Parameter:
71          * grapplerMotor: The motor used to impel the grappler
72          * touchSensor: The touchsensor is used to stop the grapplerMotor when the grappler
73          *              is in its highest position.
74          * pilot: The pilot is used for automatic close-in to the object so the grappler
    can
75          *          savely grab it.
76          */
77         Grappler2 grappler = new Grappler2(Motor.C, touch, pilot);
78
79         /**
80          * Since odometry propagates an error with each movement, a reset once in a while
    can
81          * improve over-all accuracy.
82          */
83         OdometryResetListener resetter = new OdometryResetListener();
84         Thread resetThread = new Thread(resetter);
85         resetThread.start();
86
87         /**
88          * This sensor is used by the BuoyLocator to locate objects.
89          */
90         UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S1);
91
92         /**
93          * To define a rectangle, this two points define one of its upper and one of its
    lower
94          * corners, diagonal to each other. The rectangle serves as the field.
95          * Parameter:
96          * minBorders: lower, right corner of the field. (X, Y)
97          * maxBorders: higher, left corner of the field. (X, Y)
98          */
99         Point minBorders = new Point(0,-900);
100        Point maxBorders = new Point(1800, 900);
101
102        /**
103         * The BuoyLocator is used to locate objects in the robots surroundings.
104         * Parameter:
105         * pilot: The pilot is used to rotate the robot during the searchprocess.
106         * team: The team specifies the team-membership of the robot.
107         * minBorders: The lower, right corner of the field.
108         * maxBorders: The higher, left corner of the field.
109         */
110        BuoyLocator locator = new BuoyLocator(pilot, Team.TEAMBLUE, minBorders, maxBorders,
    sonic);
111
112        /**
113         * This are searchpoints. The robot can travel to a searchpoint and locate
114         * every buoy in its surrounding.
115         * Parameter: x-y-Coordinates of the searchpoint P(x,y).
116         */
117        Point searchPoint1 = new Point(1500,400);
118        Point searchPoint2 = new Point(1000,400);
119        Point searchPoint3 = new Point(500,400);
```

```
120
121         /**
122          * This list stores the searchpoints defined above. The list-elements can be
    accessed via
123          * there index in the list. ( searchPoints.get(index); )
124          */
125         ArrayList<Point> searchPoints = new ArrayList<Point>();
126         searchPoints.add(searchPoint1);
127         searchPoints.add(searchPoint2);
128         searchPoints.add(searchPoint3);
129
130         /**
131          * This point will be used to represent the element of the list used in the loop
    below.
132          */
133         Point searchPoint;
134
135         /**
136          * This int will serve as the list-index and used to access each (one at
137          * a time) searchpoint in the loop below.
138          */
139         int searchPointsIndex = 0;
140
141         /**
142          * Wait until the user sets the startsignal by pressing any button on the robot.
143          */
144         Button.waitForAnyPress();
145
146         /**
147          * Repeat the steps in the {..} until "i >= 3"
148          */
149         while ( searchPointsIndex < 3 )
150         {
151             /**
152              * Check if the reset-button was pressed and if so,
153              * reset the pilot's odometry.
154              */
155             boolean reset = resetter.resetPlanned();
156             if (reset)
157             {
158                 pilot.DoOdometryReset();
159             }
160
161             searchPoint = searchPoints.get(searchPointsIndex);
162
163             /**
164              * Move the robot to the searchPoint defined above:
165              */
166             pilot.TravelTo(searchPoint);
167
168             /**
169              * Once arrived at the searchpoint, locate all objects in range:
170              */
171             locator.LocateObjectClockwise();
172
173             /**
174              * Check if a buoy was found:
175              */
176             boolean foundBuoy = locator.foundBuoy();
177
178             /**
179              * if a buoy was found, grab it and bring it to the base to score a point:
```

```
180             */
181            if (foundBuoy)
182            {
183                scorePoint(pilot, locator, grappler);
184            }
185
186            /**
187             * To access the next element in the list, the searchPointsIndex has to be
    iterated by one.
188             */
189            searchPointsIndex++;
190        }
191
192        /**
193         * After the every searchpoint
194         */
195        pilot.HeadHome();
196    }
197
198    /**
199     * Grabs a buoy and brings it to the base. Returns to the last position afterwards.
200     * @param pilot for movement, rotation and movement-tracking of the robot
201     * @param locator locates buoys in range
202     * @param grappler grabs and releases buoys.
203     */
204    public static void scorePoint(Pilot2 pilot, BuoyLocator locator, Grappler2 grappler)
205    {
206        // save last position:
207        Pose lastPosition = pilot.getLocation();
208
209        // close in to buoy:
210        ObjectInfo buoy = locator.getBuoyInfo();
211        locator.CloseInToBuoy(buoy);
212
213        // grab buoy
214        grappler.GrepBuoy();
215
216        // bring the buoy to your base:
217        pilot.HeadHome();
218
219        // release the buoy
220        grappler.ReleaseBuoy();
221
222        // return to last position
223        Point lastPosCoords = lastPosition.getLocation();
224        float lastHeading = lastPosition.getHeading();
225        pilot.TravelTo(lastPosCoords);
226        pilot.RotateTo(lastHeading);
227    }
228
229}
230
```

A.2.2. Vorlage 1 der Steuerung

```java
/**
 * Defines the package, this class is part of.
 * NOT IMPORTANT FOR THE WORKSOP
 */
package main;

/**
 * Import necessary libraries.
 * NOT IMPORTANT FOR THE WORKSHOP
 */
import java.util.ArrayList;

import lejos.geom.Point;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.TouchSensor;
import lejos.nxt.UltrasonicSensor;
import lejos.robotics.navigation.Pose;
import toolKit.Enums.Team;
import toolKit.ObjectInfo;
import toolKit.OdometryResetListener;
import toolKit2.BuoyLocator;
import toolKit2.Grappler2;
import toolKit2.Pilot2;

/**
 * This class represents the robot-control created as example during the Kick-Off-Presentation.
 * Every step the robot takes is defined inside this class.
 * NOTE: This is just an example and does not solve the scenario. You can use it to create
 * your own robot-control or just improve this one.
 * This robot-control uses toolKit2.
 */
public class WS_vorlage1 {
    /**
     * Main class, called and executed be the lejos-vm on your robot.
     */
    public static void main(String[] args) {
        /**
         * Variable used as parameter for the pilot, instantiated later on.
         * Wheel-diameter = 56mm, type: double.
         */
        double wheelDiameter = 56;

        /**
         * Variable used as parameter for the pilot, instantiated later on.
         * Wheel-base = 120mm, type: double.
         */


        /**
         * The pilot moves and rotates the robot and keeps track of its whereabouts.
         * Parameter:
         * wheelDiameter: The Diameter of the robot's wheels in mm.
         * wheelBase: The robot's wheelbase in mm.
         * leftMotor: The motor responsible for the left wheel (Motor.A)
         * rightMotor: The motor responsible for the right wheel (Motor.B)
         */
        Pilot2 pilot = new Pilot2(wheelDiameter, trachWidth, Motor.A; Motor.B);

        /**
         * The touchsensor is part of the grappler (instantiated later on) and shall
```

```
62          * be connected to sensorport 4 (SensorPort.S4)
63          * Parameter:
64          * port: The sensortport, the sensor is connected to (SensorPort.S4)
65          */
66
67
68         /**
69          * The grappler can grab an object like a buoy and release it afterwards.
70          * Parameter:
71          * grapplerMotor: The motor used to impel the grappler
72          * touchSensor: The touchsensor is used to stop the grapplerMotor when the grappler
73          *              is in its highest position.
74          * pilot: The pilot is used for automatic close-in to the object so the grappler
    can
75          *          savely grab it.
76          */
77
78
79         /**
80          * Since odometry propagates an error with each movement, a reset once in a while
    can
81          * improve over-all accuracy.
82          */
83         OdometryResetListener resetter = new OdometryResetListener();
84         Thread resetThread = new Thread(resetter);
85         resetThread.start();
86
87         /**
88          * This sensor is used by the BuoyLocator to locate objects.
89          */
90
91
92         /**
93          * To define a rectangle, this two points define one of its upper and one of its
    lower
94          * corners, diagonal to each other. The rectangle serves as the field.
95          * Parameter:
96          * minBorders: lower, right corner of the field. (X, Y)
97          * maxBorders: higher, left corner of the field. (X, Y)
98          * class: lejos.geom.Point
99          */
100
101
102        /**
103         * The BuoyLocator is used to locate objects in the robots surroundings.
104         * Parameter:
105         * pilot: The pilot is used to rotate the robot during the searchprocess.
106         * team: The team specifies the team-membership of the robot.
107         * minBorders: The lower, right corner of the field.
108         * maxBorders: The higher, left corner of the field.
109         * sensor: UltrasonicSensor with its SensorPort.
110         */
111
112
113        /**
114         * This are searchpoints:
115         *  P1(500,400)
116         *  P2(1000,400)
117         *  P3(1500,400)
118         * The robot can travel to a searchpoint and locate
119         * every buoy in its surrounding.
120         * Parameter: x-y-Coordinates of the searchpoint.
```

```
121          */
122
123
124         /**
125          * Once added, this list stores the searchpoints defined above. The list-elements
126          * can be accessed via there index in the list. ( searchPoints.get(index); )
127          */
128
129
130         /**
131          * This point will be used to represent the element of the list used in the loop
    below.
132          */
133
134
135         /**
136          * This int will serve as the list-index and is used to access each (one at
137          * a time) searchpoint in the loop below.
138          */
139
140
141         /**
142          * Repeat the steps in the {..} until "i >= 3"
143          */
144
145
146             /**
147              * Check if the reset-button was pressed and if so,
148              * reset the pilot's odometry.
149              */
150             boolean reset = resetter.resetPlanned();
151             if (reset)
152             {
153                 pilot.DoOdometryReset();
154             }
155
156             /**
157              * This "searchPoint" represents the elements (one at a time), accessed while
    looping
158              * over the searchpoints-list.
159              */
160
161
162             /**
163              * Move the robot to the searchPoint defined above:
164              */
165
166
167             /**
168              * Once arrived at the searchpoint, locate all objects in range:
169              */
170
171
172             /**
173              * Check if a buoy was found:
174              */
175
176
177             /**
178              * if a buoy was found, grab it and bring it to the base to score a point:
179              */
180
```

```
181
182             /**
183              * To access the next element in the list, the index has to be iterated by one.
184              */
185
186
187
188         /**
189          * After the every searchpoint
190          */
191
192     }
193
194     /**
195      * Grabs a buoy and brings it to the base. Returns to the last position afterwards.
196      * @param pilot for movement, rotation and movement-tracking of the robot
197      * @param locator locates buoys in range
198      * @param grappler grabs and releases buoys.
199      *
200      * steps:
201      *  - save last position
202      *  - close in to buoy
203      *  - grab buoy
204      *  - bring buoy to base
205      *  - release buoy
206      *  - return to last position
207      *  - rotate to last heading
208      */
209
210
211 }
212
```

A.2.3. Vorlage 2 der Steuerung

```java
/**
 * Defines the package, this class is part of.
 * NOT IMPORTANT FOR THE WORKSOP
 */
package main;

/**
 * Import necessary libraries.
 * NOT IMPORTANT FOR THE WORKSHOP
 */
import java.util.ArrayList;

import lejos.geom.Point;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.TouchSensor;
import lejos.nxt.UltrasonicSensor;
import toolKit.Enums.Team;
import toolKit.OdometryResetListener;
import toolKit2.BuoyLocator;
import toolKit2.Grappler2;
import toolKit2.Pilot2;

/**
 * This class represents the robot-control created as example during the Kick-Off-Presentation.
 * Every step the robot takes is defined inside this class.
 * NOTE: This is just an example and does not solve the scenario. You can use it to create
 * your own robot-control or just improve this one.
 */
public class WS_vorlage2 {
    /**
     * Main class, called and executed be the lejos-vm on your robot.
     */
    public static void main(String[] args) {
        /**
         * Variable used as parameter for the pilot, instantiated later on.
         * Wheel-diameter = 56mm.
         */
        double wheelDiameter = 56.0;

        /**
         * Variable used as parameter for the pilot, instantiated later on.
         * Wheel-base = 120mm
         */
        double wheelBase = 120.0;

        /**
         * The pilot moves and rotates the robot and keeps track of its whereabouts.
         * Parameter:
         * wheelDiameter: The Diameter of the robot's wheels in mm.
         * wheelBase: The robot's wheelbase in mm.
         * leftMotor: The motor responsible for the left wheel (Motor.A)
         * rightMotor: The motor responsible for the right wheel (Motor.B)
         */
        Pilot2 pilot = new Pilot2(wheelDiameter, wheelBase, Motor.A, Motor.B);

        /**
         * The touchsensor is part of the grappler (instantiated later on) and shall
         * be connected to sensorport 4 (SensorPort.S4)
         * Parameter:
         * port: The sensortport, the sensor is connected to (SensorPort.S4)
```

```
62          */
63         TouchSensor touch = new TouchSensor(SensorPort.S4);
64
65         /**
66          * The grappler can grab an object like a buoy and release it afterwards.
67          * Parameter:
68          * grapplerMotor: The motor used to impel the grappler
69          * touchSensor: The touchsensor is used to stop the grapplerMotor when the grappler
70          *              is in its highest position.
71          * pilot: The pilot is used for automatic close-in to the object so the grappler
    can
72          *          savely grab it.
73          */
74         Grappler2 grappler = new Grappler2(Motor.C, touch, pilot);
75
76         /**
77          * Since odometry propagates an error with each movement, a reset once in a while
    can
78          * improve over-all accuracy.
79          */
80         OdometryResetListener resetter = new OdometryResetListener();
81         Thread resetThread = new Thread(resetter);
82         resetThread.start();
83
84         /**
85          * This sensor is used by the BuoyLocator to locate objects.
86          * SensorPort.S1 to SensorPort.S4
87          */
88         UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S1);
89
90         /**
91          * To define a rectangle, this two points define one of its upper and one of its
    lower
92          * corners, diagonal to each other. The rectangle serves as the field.
93          * Parameter:
94          * minBorders: lower, right corner of the field. (X, Y)
95          * maxBorders: higher, left corner of the field. (X, Y)
96          */
97         Point minBorders = new Point(0,-900);
98         Point maxBorders = new Point(1800, 900);
99
100        /**
101         * The BuoyLocator is used to locate objects in the robots surroundings.
102         * Parameter:
103         * pilot: The pilot is used to rotate the robot during the searchprocess.
104         * team: The team specifies the team-membership of the robot.
105         * minBorders: The lower, right corner of the field.
106         * maxBorders: The higher, left corner of the field.
107         */
108        BuoyLocator locator = new BuoyLocator(pilot, Team.TEAMBLUE, minBorders, maxBorders,
    sonic);
109
110        /**
111         * This are searchpoints. The robot can travel to a searchpoint and locate
112         * every buoy in its surrounding.
113         * Parameter: x-y-Coordinates of the searchpoint P(x,y).
114         */
115        Point searchPoint1 = new Point(2000,400);
116        Point searchPoint2 = new Point(1000,400);
117
118
119        /**
```

```
120          * This list stores the searchpoints defined above. The list-elements can be
    accessed via
121          * there index in the list. ( searchPoints.get(index); )
122          */
123         ArrayList<Point> searchPoints = new ArrayList<Point>();
124         searchPoints.add(searchPoint1);
125         searchPoints.add(searchPoint2);
126 
127         /**
128          * This point will be used to represent the element of the list used in the loop
    below.
129          */
130         Point searchPoint;
131 
132         /**
133          * This int will serve as the list-index and used to access each (one at
134          * a time) searchpoint in the loop below.
135          */
136         int searchPointsIndex = 0;
137 
138         /**
139          * Repeat the steps in the {..} until "i >= 3"
140          */
141         while ( searchPointsIndex < 3 )
142         {
143             /**
144              * Check if the reset-button was pressed and if so,
145              * reset the pilot's odometry.
146              */
147             boolean reset = resetter.resetPlanned();
148             if (reset)
149             {
150                 pilot.DoOdometryReset();
151             }
152 
153             searchPoint = searchPoints.get(searchPointsIndex);
154 
155             /**
156              * Move the robot to the searchPoint defined above:
157              */
158             pilot.TravelTo(searchPoint);
159 
160             /**
161              * Once arrived at the searchpoint, locate all objects in range:
162              */
163             locator.LocateObjectClockwise();
164 
165             /**
166              * To access the next element in the list, the searchPointsIndex has to be
    iterated by one.
167              */
168             searchPointsIndex++;
169         }
170 
171         /**
172          * After the every searchpoint
173          */
174         pilot.HeadHome();
175     }
176 
177     /**
178      * Grabs a buoy and brings it to the base. Returns to the last position afterwards.
```

```
179      * @param pilot for movement, rotation and movement-tracking of the robot
180      * @param locator locates buoys in range
181      * @param grappler grabs and releases buoys.
182      */
183     public static void scorePoint(Pilot2 pilot, BuoyLocator locator, Grappler2 grappler)
184     {
185         // save last position:
186 
187 
188         // close in to buoy:
189 
190 
191         // grab buoy
192 
193 
194         // bring the buoy to your base:
195 
196 
197         // release the buoy
198 
199 
200         // return to last position
201 
202     }
203 
204 }
205 
```

A.2.4. Vorlage für die Spezifikation

Spezifikation
LEGO-Mindstorms Workshop

Team: _____

Teilnehmer: _____

_____

_____

Notizen:

A.3. Handout

Überblick: Java-Sprachelemente

Schleifen:

```
while (<BEDINGUNG>)
{
    // Anweisung
}
```

```
do
{
    // Anweisung
}
while (<BEDINGUNG>)
```

if- / else:

```
if (<BEDINUNG>)
{
    // Anweisung im Fall <BEDINGUNG> ist wahr
}
else
{
    // Anwesiung im anderen Fall, <BEDINGUNG> ist falsch
}
```

Typen, Objekte und Rückgabewerte:

```
int:        int varInt = 0;             // Alle ganzen Zahlen
float:      float varFloat = - 4.0;     // Alle Kommazahlen
double:     double varDouble = -4.0;    // Wie „float“
boolean:    boolean varBool = true;     // {true, false}, für Bedinungen

void;       // kein Typ! Beschreibt „kein Rückgabewert“ bei Methoden.

Klasse      Objektbezeichner        Aufruf Konstruktor
Pilot2      pilot                   = new Pilot2 (<Parameter>);
```

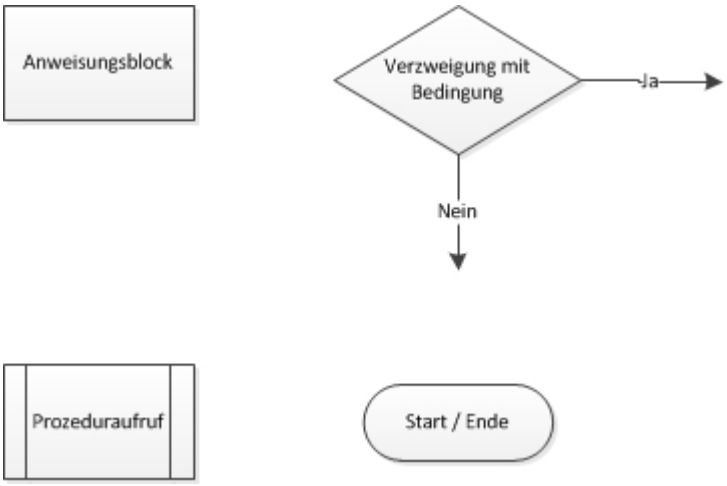
Methoden:

```
public static <RückgabeTyp> Bezeichner (<Parameter>)
{
    // Anweisung(en)
}
```

<u>Aufrufbeispiel:</u>

```
int result = Multiply(2, 6)     // Multipliziert die beiden Parameter und
                                // schreibt das Ergebnis in die Variable „result“.
```

Programmablaufnotation:



Das Szenario

- Auf dem Spielfeld sind verschiedenfarbige Bojen verteilt.
- Die Bojen sind entsprechend ihrer Farbe unterschiedlich viele Punkte wert.
- Punkte gibt es nur für erfolgreich zur eigenen Basis gebrachte Bojen.

Literaturverzeichnis

[1] Castor-bot. http://www.nxtprograms.com/castor_bot/index.html. (Zitiert auf Seite 17)

[2] Generation robots. http://www.generationrobots.com/ultrasonic-sonar-sensors-for-robots,us,8,19.cfm. (Zitiert auf den Seiten 8, 50 und 51)

[3] Institute of electrical and electronics engineers (ieee). www.ieee.org. (Zitiert auf Seite 11)

[4] Studiengang an der uni-stuttgart. http://www.uni-stuttgart.de/studieren/angebot/software_bsc/. (Zitiert auf Seite 9)

[5] Studistatistiken an der uni-stuttgart. http://www.uni-stuttgart.de/ueberblick/wir_ueber_uns/zahlen_fakten/statistik/studstat.html. (Zitiert auf Seite 9)

[6] Lejos - java for lego mindstorms, 2006 to 2012. http://lejos.sourceforge.net/. (Zitiert auf den Seiten 21, 26 und 36)

[7] Eclipse-Foundation. Eclipse ide for java-developement. www.eclipse.org/. (Zitiert auf den Seiten 21 und 26)

[8] Lego. Lego. http://www.lego.com/. (Zitiert auf Seite 9)

[9] Lego. Lego-mindstorms. http://mindstorms.lego.com/eng/default.aspx. (Zitiert auf den Seiten 9, 15 und 23)

[10] Lego. Nxt software. http://service.lego.com/en-us/helptopics/?questionid=2655. (Zitiert auf Seite 15)

[11] lejos. lejos - api. http://lejos.sourceforge.net/nxt/nxj/api/index.html. (Zitiert auf Seite 21)

[12] lejos. Lejos - plugin für eclipse. http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/UsingEclipse.htm. (Zitiert auf Seite 21)

[13] lejos. lejos - wiki. http://sourceforge.net/apps/mediawiki/lejos/index.php?title=Main_Page. (Zitiert auf Seite 21)

[14] Jochen Ludewig and Horst Lichter. *Software Engineering*. dpunkt.verlag. 2., überarbeitete, aktualisierte u. ergänzte Auflage. (Zitiert auf den Seiten 11, 12 und 13)

[15] Mozilla. Mozilla-puplic-license (mpl). http://www.mozilla.org/MPL/. (Zitiert auf Seite 21)

[16] NXT-Programs. Fun projects for your lego mindstorms nxt! http://www.nxtprograms.com/index.html. (Zitiert auf Seite 17)

Alle URLs wurden zuletzt am 23. 01. 2013 geprüft.

Erklärung:

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst habe und dabei keine andere als die angegebene Literatur verwendet habe. Alle Zitate und sinngemäßen Entlehnungen sind als solche unter genauer Angabe der Quelle gekennzeichnet.

_____

(Raimund Metzler)