KOOPERATIVE STEUERUNG VON MODELLVERSUCHSFAHRZEUGEN

ENTWICKLUNG EINES INTELLIGENTEN FAHRENTSCHEIDERS

Softwareentwicklungspraktikum
Sommersemester 2008

Feinentwurf zum System

E C A R

Auftraggeber
Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Prof. Dr.-Ing. Lars Wolf
Mühlenpfordstrasse 23
38106 Braunschweig

Betreuer: Kai Homeier, Carina Flämig
Phasenverantwortlicher: Ekrem Enes Duman

Auftragnehmer

Name	E - Mail
Wira Kakar	wirak@web.de
Chen Zhiwei	czwnii@hotmail.com
Rayan Merched El Masri	rayan_masri@hotmail.com
Ekrem Enes Duman	dumanenes@hotmail.de
Günter Dusch	g.dusch@Googlemail.com

Braunschweig, 23.05.2008

Versionsübersicht

Version	Datum	Autor	Status	Kommentar
1.0	17.05.08	Alle Auftragsnehmer		
2.0	23.05.08	Alle Auftragsnehmer		Korrektur: Klassendiagramme, neue Methoden der Bildverarbeitung, Positionierung und Wegentscheidung, Datenmodell und Verbesserung hinsichtlich Sprache, Grammatik und Rechtschreibung

Inhaltsverzeichnis

A b b i l d u n g s v e r z e i c h n i s

1 Einleitung

Im vorliegenden Feinentwurf werden die bereits in dem Pflichtenheft und im Grobentwurf thematisierten Inhalte, die Konstruktion eines Softwaresystems zur autonomen Steuerung eines Modellversuchsfahrzeugs, weiter konkretisiert. Dies geschieht mit dem Ziel, dass dieses Dokument es jedem externen Programmierer ermöglicht die Problematik zu erfassen und insbesondere die Umsetzung anhand dieser Vorlage problemlos zu ermöglichen. Somit werden programmspezifische Details und technische Elemente im Fokus stehen.
Dafür werden zu aller erst die elementaren Bedingungen, die Musskriterien und im Anschluss die Wunschkriterien behandelt. Eine erste Antwort auf die Frage, wie diese umzusetzen sind, wird hier gegeben. Das „Herzstück“ dieses Dokuments folgt darauf in dem Abschnitt Implementierungsentwurf, in dem die verwendeten Klassen und Bibliotheken mit den entsprechenden Attributen und Methoden, unter anderem mit der Zuhilfenahme von Klassendiagrammen sowie anderen Darstellungsformen der Unified Modeling Language (UML), dokumentiert werden.

2 Erfüllung der Kriterien

2.1 Musskriterien

/M10/ Die Identifikation von Baken zur Navigation wird anhand der Klasse „Search“, mit Hilfe der Methoden „seeBeacon“, „newBeacon“, und „distaceToBeacon“, realisiert.

/M20/ Das gesuchte Objekt wird anhand der Methode „searchBall“ der Klasse „Search“ erkannt bzw. identifiziert.

/M30/ Wie sich das Modellversuchsfahrzeug innerhalb des Labyrinths fortzubewegen hat, wird mithilfe der Klasse „Decision“ realisiert. Dafür sind die Methoden „decideAngle“ und „decideSegment“ vorgesehen.

/M40/ Die Speicherung der bereits besuchten Wege wird mit der Methode „visitedWay“ der Klasse „Labyrinth“ realisiert.

/M50/ Das Auffinden des gesuchten Objekts kann nur realisiert werden, wenn die Erfüllung aller Musskriterien erfolgt.

/M60/ Nachdem der Winkel mit den Baken gemessen wurde, erfolgt die Positionsberechnung mit der Methode „position“. Die Position wird in bestimmtem Zeitabstand berechnet oder wenn das nötig ist.

2.2 Wunschkriterien

/W10/ Erstellung einer Karte des Labyrinths:

Hier werden sowohl die Hindernisse gezeichnet als auch die bereits besuchten Wege visuell in Kartenform (maßstabsgetreu) dargestellt.

/W20/ Auffinden des gesuchten Objektes in minimaler Zeit:

Für dieses Wunschkriterium ist die Erstellung einer Karte notwendig. Notwendig sind aber auch die Verfügbarkeit einer Kamera sowie die von Abstandssensoren, die eine angemessene Reichweite besitzen.

/W30/ Effizienzsteigerung durch den Einsatz mehrerer autonomer Fahrzeuge:

Die Fahrzeuge sollen miteinander (drahtlos) kommunizieren, indem Karten mit Informationen über die Wege ausgetauscht werden. Die einzelnen Fahrzeuge erstellen ihre eigenen Karten des Labyrinths, wobei diese unvollständigen Karten zwischen den Fahrzeugen ausgetauscht und in einer neuen Karte zusammenzufügen sind.

3 Implementierungsentwurf

3.1 Gesamtsystem

Dieses Komponentendiagramm veranschaulicht die wesentlichen Komponenten des Systems mit ihren jeweiligen Kommunikationsbeziehungen untereinander. Das Softwaresystem ECAR basiert auf drei elementaren Säulen. Die erste Komponente ist die „Bildverarbeitung“, die zweite die „Positionierung“ des Modellversuchs-fahrzeugs und die dritte Komponente wird durch die „Wegentscheidung“ dargestellt.
Für die Positionierung des Modellversuchsfahrzeugs innerhalb des Labyrinths ist es eine Notwendigkeit die Kommunikation zwischen der ersten Komponente, der „Bildverarbeitung“ und der zweiten Komponente, der „Positionierung“ zu ermöglichen, da die Positionsberechnung erst durchgeführt werden kann, wenn aufgrund des visuellen Sensors die für die Navigation nötigen Baken erkannt werden. Anschließend kann sich das Modellversuchsfahrzeug erst erfolgreich positionieren. Somit ist es offensichtlich wieso hier eine Schnittstelle bestehen muss.

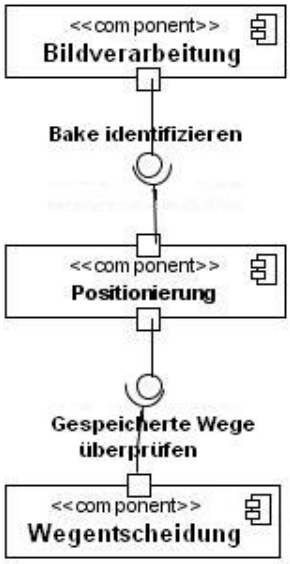

Abbildung 1: Komponentendiagramm

Die Abhängigkeitsbeziehungen zwischen der Komponente zwei, der „Positionierung“ und der Komponente drei, die „Wegentscheidung“ sind äquivalent zu den vorigen beiden. Das bedeutet, dass die Komponente Wegentscheidung die erfolgreiche Informationsübermittlung über eine Schnittstelle von der Komponente „Positionierung“ voraussetzt, um eine angemessene Richtungswahl durchführen zu können, da die Richtungswahl, die zur Wegentscheidung führt wiederum als Bedingung den Abgleich mit den bereits besuchten Wegen hat und das nur mit dem Wissen über die bereits besuchten Wege von der Komponente „Positionierung“ möglich ist. Sonst würden Wege mehrfach befahren werden, was einer ineffizienten Vorgehensweise entspricht. Daher muss zwischen den beiden Komponenten eine Schnittstelle bestehen.

3.2 Implementierung von Komponente Bildverarbeitung

3.2.1 Klassendiagramm

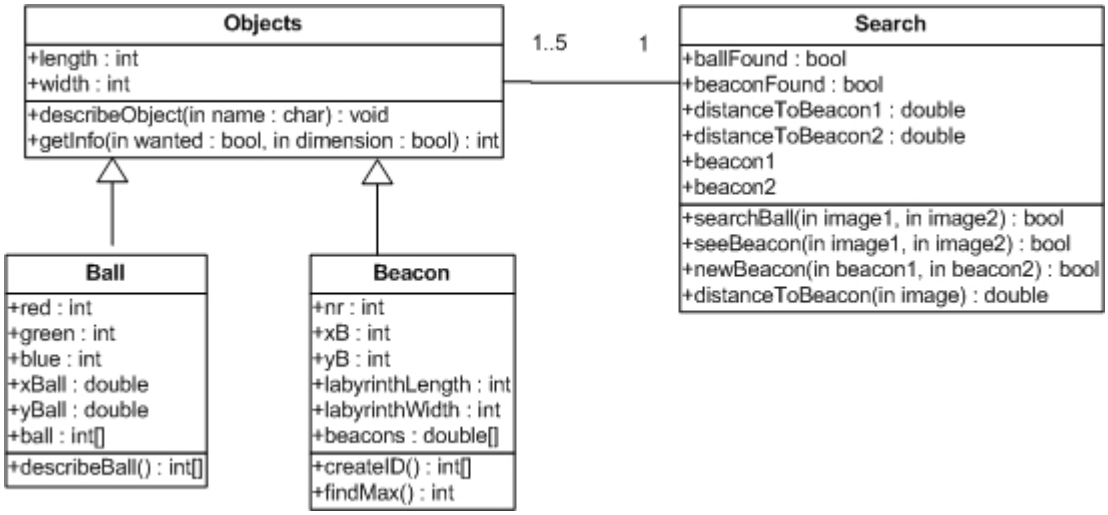

Abbildung 2: Implementierung der Komponente Bildverarbeitung

3.2.2 Erläuterung

Klasse Objekts

Attribute:

- length: Hier wird die Länge der Baken und des Balls gespeichert.
- width: Hier wird die Breite der Baken und des Balls gespeichert. Da der Ball einen Radius hat, sind die Länge und die Breite des Balls gleich groß.

Methoden:

- ***/100/*** describeObjekt: Diese Methode soll bei der Beschreibung der Baken und des Balls helfen. Wenn man den Ball beschreiben will, gibt man einfach „Ball“ ein und die Methode „describeBall“ wird aufgerufen. Will man die Baken beschreiben, wird „Beacon“ eingegeben und die Methode „createID“ wird aufgerufen.

Kommunikationspartner:

- Ball: Keine dauerhafte Kommunikation. Die Kommunikation ist erforderlich, wenn der Ball beschrieben werden soll und die Methode „searchBall“ die Daten des Balls abfragt.
- Beacon: Keine dauerhafte Kommunikation Die Kommunikation ist erforderlich, wenn die Baken beschrieben werden sollen und die Methode „seeBeaconl“ die Daten der Baken abfragt.
- Search: Dauerhafte Kommunikation. Da die Methoden der Klasse „Search“ immer die Baken oder den Ball suchen ist eine dauerhafte Kommunikation notwendig.
- Labyrinth: Keine dauerhafte Kommunikation. Die Kommunikation ist erforderlich wenn die Methode „divide“ die Werte „maxLength“ und „maxWidth“ bestimmt.

Klasse Beacon

Attribute:

- nr: wird als Index für das Feld beacons[ ] verwendet.
- xB: x-Koordinate der Bake.
- yB: y-Koordinate der Bake.
- labyrinthLength: Hier wird die Länge des Labyrinths gespeichert.
- labyrinthWidth: Hier wird die Breite des Labyrinths gespeichert.
- beacons: In diesem Feld werden die jeweilige ID´s der Baken gespeichert. Dieses Feld hat die Länge 4.

Methoden:

- **/110/** createID: Diese Methode erstellt eine ID für jede Bake. Diese ID besteht aus den beiden Dimensionen (Länge und Breite) und die Position der Bake.
  Die ID ist eine 10-stellige Zahl wobei die ersten drei Stellen für die x- und die darauf folgenden drei für die y-Koordinate stehen. Die vierte und dritte Stelle werden durch die Länge repräsentiert und die letzten beiden spiegeln die Breite der Bake wieder. Eine ID könnte so aussehen: xxxyyyllww.

Die ID´s werden in dem Feld beacons[ ] gespeichert und bei Bedarf können die Dimensionen und Positionen durch die Division- und Moduloperationen berechnet werden.

- **/111/** findMax: Diese Methode berechnet die Länge und die Breite des Labyrinths, wobei die einzelne Koordinaten der Baken miteinander verglichen werden. Die größte x-Koordinate wird in labyrinthLength und die größte y-Koordinate in labyrinthWidth gespeichert.

Kommunikationspartner:

- Objects: Keine dauerhafte Kommunikation

Klasse Ball

Attribute:

- red: Stellt die Rot-Komponente der Farbe des Balls dar.
- green: Stellt die Grün-Komponente der Farbe des Balls dar.
- blue: Stellt die Blau-Komponente der Farbe des Balls dar.
- xBall: Am Anfang wird dieses Attribut mit -1 initialisiert. Nachdem der Ball gefunden wurde, wird hier die x-Koordinate des Balls gespeichert.
- yBall: Am Anfang wird dieses Attribut mit -1 initialisiert. Nachdem der Ball gefunden wurde, wird hier die y-Koordinate des Balls gespeichert.
- ball: In diesem Feld werden die Farbkomponenten des Balls gespeichert.

Methoden:

- **/120/** describeBall: Mit dieser Methode wird der Ball beschrieben. Die Werte „red“-, „green“- und „blue“ sowie „length“ werden in dem Feld „ball[ ]“ gespeichert.

Kommunikationspartner:

- Objects: Keine dauerhafte Kommunikation.

Klasse Search

Attribute:

- ballFound: Dieses Attribut ist ein boolescher Wert. Am Anfang wird es mit false initialisiert und erst dann auf true gesetzt, wenn der Ball gefunden wurde.
- beaconFound: Dieses Attribut ist ein boolescher Wert. Am Anfang wird es mit false initialisiert und erst dann auf true gesetzt, wenn eine Bake gefunden wurde.
- distanceToBeacon1: Hier wird der Abstand zwischen dem Modellversuchs-fahrzeug und der ersten Bake gespeichert.
- distanceToBeacon2: Hier wird der Abstand zwischen dem Modellversuchs-fahrzeug und der zweiten Bake gespeichert.
- beacon1: Das Bild der ersten Bake wird hier gespeichert.
- beacon2: Das Bild der zweiten Bake wird hier gespeichert.

Methoden:

- **/130/** searchBall: Sucht nach dem Ball, indem das aufgenommene Bild mit dem Bild vom Ball verglichen wird (mit Hilfe von OpenCV-Funktionen). Wenn der Ball gefunden wird, setzt diese Methode den Wert „ballFound“ auf true.
- **/131/** seeBeacon: Vergleicht das aufgenommene Bild mit den Bildern der Baken, die in Klasse „Beacon“ beschrieben wurden. Solange es keine Bake gefunden hat, setzt es den Wert „beaconFound“ auf false, sonst auf true.
- **/132/** newBeacon: Diese Methode vergleicht zwei gefundene Baken miteinander. Wenn die beide gleich sind, wird nach einer neuen zweiten Bake weiter gesucht.
- **/133/** distanceToBeacon: Hier wird der Abstand zwischen dem Modell-versuchsfahrzeug und der Bake berechnet.

Kommunikationspartner:

- Objects: Dauerhafte Kommunikation. Alle Methoden der Klasse „Search “ suchen nach Objekten (Baken oder Ball), deren Beschreibungen sich in der Klasse „Objects“ befinden.

- Position: Keine dauerhafte Kommunikation. Die Kommunikation ist erforderlich, wenn der Suchvorgang der Software hinsichtlich der Baken stattfindet.

3.3 Implementierung von Komponente Positionierung

3.3.1 Klassendiagramm

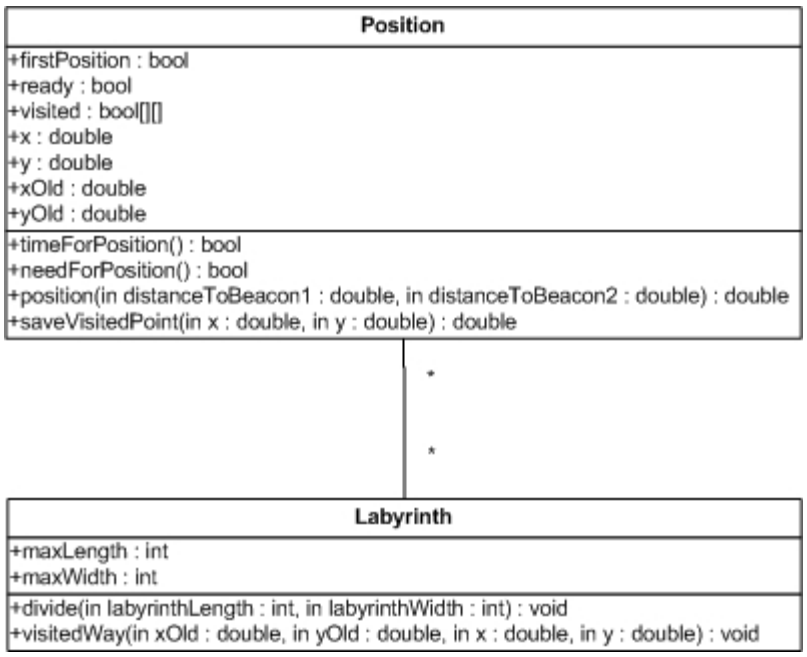

Abbildung 3: Implementierung der Komponente Positionierung

3.3.2 Erläuterung

Klasse Position

Berechnet und speichert die Position des Modellversuchsfahrzeugs. Auch wird nach spezifisch festgelegten Intervallen (Timer) oder nach einer Richtungsänderung die Position des Fahrzeugs neu bestimmt. Den Variablen „xOld“, „yOld“ werden die Koordinaten des ersten Punktes, an dem eine Richtungsänderung oder Positionsberechnung stattfindet, zugewiesen. Das Gleiche geschieht mit den beiden Variablen x und y bei der darauf folgenden Richtungsänderung, so dass zwei Punkte mit den jeweiligen Koordinaten bekannt sind und so der zurückgelegte Weg als Gerade zwischen diesen beiden Punkten berechnet werden kann.

Attribute:

- firstPosition: Dieses Attribut hat am Anfang den Wert true. Nach dem die erste Position berechnet wurde, bekommt das Attribut den Wert false, was zur Folge hat, dass jetzt die Methode „visitedWay“ durchgeführt werden kann. Wenn der Wert des Attributs true ist, werden die aktuellen Koordinaten des Fahrzeugs in „xOld“ und „yOld“ gespeichert. Danach wird der Wert auf false gesetzt. Nach der Berechnung der zweiten Position wird die Methode „visitedWay“ aufgerufen.
- ready: Ziel dieses Attributes ist, dass das Fahrzeug die Position bestimmt. Bei der Initialisierung hat dieses Attribut den Wert false. Die Methoden „timeForPosition“ und „needForPosition“ setzen den Wert dieser Variablen auf true. Erst wenn der Wert dieses Attributes true ist kann die Methode „position“ ausgeführt werden.
- visited: Dieses zweidimensionale Array repräsentiert das in kleine Quadranten zerlegte Labyrinth. Am Anfang sind alle Array-Elemente auf false gesetzt. Wenn sich das Fahrzeug innerhalb des Labyrinths fortbewegt, dann durchfährt es eine Schnittmenge dieser Quadranten, die dann den Wert true zugewiesen bekommen, da sie ja besucht wurden. Jeder Quadrant ist eindeutig identifizierbar anhand einer Nummer, die sich aus der Größe des Gesamtlabyrinths ergibt. Ein Quadrant ist 20 cm x 20 cm groß. Dies ist die Art und Weise wie sichergestellt wird, dass das Fahrzeug zwischen besuchten und unbesuchten Wegen unterscheiden kann.
- x: Dieser Variablen wird die X-Koordinate des zweiten Punktes zugewiesen.
- y: Dieser Variablen wird die Y-Koordinate des zweiten Punktes zugewiesen.
- xOld: Dieser Variablen wird die X-Koordinate des ersten Punktes zugewiesen.
- yOld: Dieser Variablen wird die Y-Koordinate des ersten Punktes zugewiesen.

Methoden:

- **/200/** timeForPosition: Diese Methode enthält einen Timer, der einen Countdown durchführt, wonach der boolesche Wert true der Variablen „ready“ zugewiesen wird.

- **/201/** needForPosition: Wenn das Fahrzeug die Richtung ändert, wird „ready“ auf true gesetzt. D.h. dass die Position immer dann berechnet wird, wenn entweder der Timer abläuft oder eine Richtungsänderung stattfindet.
- **/202/** position: Berechnet die Position des Fahrzeugs mit der mathematischen Methode der Kreuzpeilung und gibt diese als x- und y-Koordinaten zurück.
- **/203/** saveVisitedPoint: Die Methode wird benötigt um den letzten besuchten Punkt zwischen zu speichern. Am Anfang ist das Attribut „firstPosition“ auf true gesetzt, wenn der erste Punkt noch nicht berechnet wurde. Um die zurückgelegte Strecke berechnen zu können benötigt man zwei Punkte. Der erste Punkt wird in der Variablen „xOld“ und „yOld“, und die Koordinaten des zweiten Punkt in „x“ und „y“ gespeichert. Diese Methode dient als eine Zwischenspeicherung für die besuchte Koordinaten.

Kommunikationspartner:

- Labyrinth: Dauerhafte Kommunikation. Nach jeder Berechnung eines Weges werden die besuchten Quadranten durch die Methode „saveVisited“ gespeichert.

Klasse Labyrinth

Attribute:

- maxLength: Der Wert dieses Attributs gibt die Länge des Labyrinths wieder.
- maxWidth: Der Wert dieses Attributs gibt die Länge des Labyrinths wieder.

Methode:

- **/210/** divide: Diese Methode unterteilt das Labyrinth in gleich große Quadranten, welche auch einen Index erhalten und somit eindeutig identifizierbar sind. Sie berechnet die Maße des Labyrinths, da die Positionen der Baken bekannt sind und diese an den vier Eckpunkten des rechteckigen Labyrinths aufgestellt sind. Die Größe der Quadranten ergibt sich aus der Division der X- und Y-Achse des Labyrinths mit der Konstante 20. Die Ergebnisse der Division (Länge / 20 und Breite / 20) werden den Attributen „maxLength“ und „maxWidth“ zugewiesen. Nach dieser Berechnung wird das

zweidimensionale Feld „visited“ mit den Werten false initialisiert. Dabei zählt die erste Dimension von Null bis zu dem Wert der Variablen „maxLength“ und die zweite Dimension von Null bis zu dem Wert des Attributs „maxWidth“ (visited[i][j] i=0 bis maxLength und j=0 bis maxWidth).

- **/211/** visitedWay: Die Methode bekommt die Koordinaten des ersten und zweiten Punktes (Ort der Richtungsänderung) und berechnet dann die Verbindungsgerade zwischen den beiden Punkten. Die Quadranten, die von dieser Geraden geschnitten werden, werden als besucht markiert, indem das Attribut „visited“ auf „true“ gesetzt wird.

3.4 Implementierung von Komponente Wegentscheiden

3.4.1 Klassendiagramm

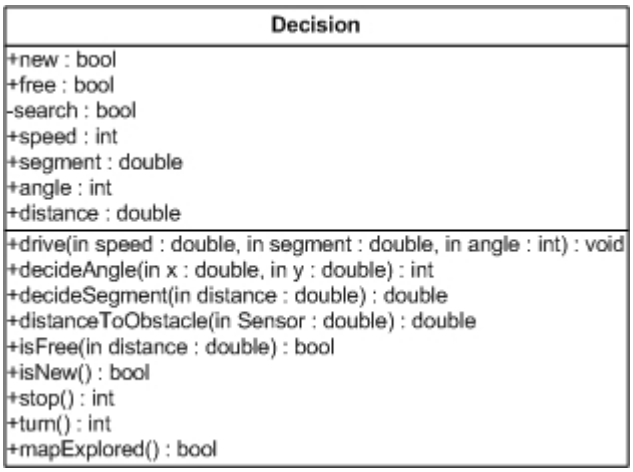

Abbildung 4: Implementierung der Komponente Wegentscheiden

3.4.2 Erläuterung

Klasse Decision

Attribute:

- new: Der Wert dieses Attributs ist am Anfang false. Wenn aber ein Weg ein zweites Mal besucht wird, ändert sich der Rückgabewert dieses Attributs auf true.
- free: Wenn der von den Sensoren gemessene Abstand zu einem Hindernis einen gewissen Wert unterschreitet, dann hat dieses Attribut false als Rückgabewert. Wenn allerdings dieser Mindestabstand eingehalten wird, dann hat das Attribut den Wert true.
- search: Solange der Wert true dem Attribut „search“ zugewiesen ist, wird der Suchvorgang fortgesetzt. Sobald der Suchvorgang beendet wird, weil der Ball gefunden oder alle Wege abgesucht wurden, ändert sich der Wert von „search“ auf false.

- speed: Die Geschwindigkeit des Fahrzeugs wird in dem Attribut „speed“ als Double-Wert gespeichert.
- segment: Dem Fahrzeug wird z.B. die Anweisung gegeben eine 50 cm Strecke zu fahren. Diese 50 cm lange Strecke ist das Attribut „segment“. Segment ist auch in der Funktion „speed“ wiederzufinden.
- angle: Angle ist ein Winkel, in dessen Bereich sich das Fahrzeug bewegt.
- distance: Distance enthält Double-Werte, die die gemessenen Werte der Sensoren widerspiegeln. Die Sensoren messen Abstände zu Hindernissen in Bit-Werten. Diese Bit-Werte werden von der Methode „distanceToObstacle“ in die Einheit Zentimeter umgewandelt. Die Zentimeter-Werte sind nun in distance abgelegt.

Methoden:

- **/300/** drive: Diese Methode manipuliert die Geschwindigkeit und die Richtung (Winkel) des Modellversuchsfahrzeugs.
- **/301/** decideAngle: Diese Methode sorgt für die Entscheidung, in welche Richtung sich das Fahrzeug bewegt. Bei dieser Methode ist die Position des Fahrzeugs der Eingabewert. Der Rückgabewert ist ein Int-Wert, der in das Attribut Angle eingesetzt wird.
- **/302/** decideSegment: Die Wahrnehmung der Umwelt des Fahrzeugs über die Sensoren wird an diese Methode übergeben, die anschließend den Halte-zeitpunkt (wie lange das Fahrzeug fahren darf) berechnet und durchführt. Dies wird im Attribut „segment“ abgelegt.
- **/303/** distanceToObstacle: Hier werden die 8-Bit-Werte der Sensoren in cm umgewandelt, die in das Attribut distance abgelegt werden.
- **/304/** isFree: Die Werte der Sensoren werden von der Methode „isFree“ analysiert, um herauszufinden, ob der Weg frei ist oder ein Hindernis im Weg steht. Wenn der Abstand für eine angemessene Fortbewegung groß genug ist, wird das Attribut free auf true gesetzt. Wenn aber der Abstand zu klein ist, so dass sich das Fahrzeug in diesem Bereich nicht bewegen kann, wird dem Attribut free der Wert false zugewiesen.

- **/305/** isNew: Diese Funktion untersucht, ob ein Weg bereits besucht wurde. Falls dieser Weg bereits besucht wurde, wird das Attribut „new“ auf true gesetzt, sonst false.
- **/306/** stop: Diese Methode gibt dem Fahrzeug den Befehl zu stoppen, d.h. dass das Attribut „speed“ gleich Null gesetzt wird. Dieser Vorgang tritt auf, wenn das Fahrzeug auf einen bereits besuchten Weg trifft, ein Hindernis sieht oder das gesuchte Objekt findet.
- **/307/** turn: Die Methode „turn“ sorgt dafür, dass das Fahrzeug seine Fahrtrichtung ändert, d.h. der Winkel geändert wird.
- **/308/** mapExplored: „mapExplored“ überprüft die Informationen der Methode visitedWay aus der Klasse Position, ob alle Wege und Stellen im Labyrinth besucht worden sind. Wenn alle Wege im Labyrinth abgefahren wurden, setzt die Methode den Wert von „search" auf false, sonst auf true.

Kommunikationspartner

- Labyrinth: Keine dauerhafte Kommunikation. Mit der Methode „mapexplored“ fragt die Klasse „Decision“ den Feld „visited[ ][ ]“ der Klasse Labyrinth ab, um festzustellen ob alle Wege besucht sind oder nicht.

4 Datenmodell

4.1 Diagramm

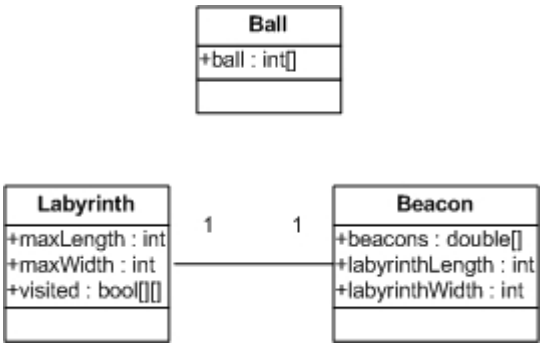

Abbildung 5: Datenmodell

4.2 Erläuterung

Die Felder „ball[ ]“ und „beacons[ ]“ enthalten die Daten des Balls bzw. der Baken, die sich mit der Zeit nicht ändern und dauerhaft von mehreren Methoden abgefragt werden.

Für die Erstellung der Karte, sind die Länge und Breite des Labyrinths erforderlich. Diese Werte (LabyrinthLength und LabyrinthWidth) werden für die Berechnung der Werte „maxLength“ und „maxWidth“ verwendet. Im Feld „visited[ ]“ werden die bereits besuchten Quadranten gespeichert.