Validierungsbericht

Phase	Verantwortlicher	E-Mail
Pflichtenheft	Florian Lorenz	lorenz@fim.uni-passau.de
Entwurf	Andreas Wilhelm	wilhelma@fim.uni-passau.de
Spezifikation	Andreas Poxrucker	poxrucke@fim.uni-passau.de
Implementierung	Martin Freund	freund@fim.uni-passau.de
Validierung	Florian Bürchner	buerchne@fim.uni-passau.de
Präsentation	Max Binder	binder@fim.uni-passau.de

21. Januar 2011

Inhaltsverzeichnis

1 Globale Testszenarien und Testfälle **3**
1.1 Pflichtenheft Testfälle . . . 3
1.1.1 /T50/ Kalibrierung . . . 3
1.1.2 /T60/ Linienerkennung . . . 3
1.1.3 /T70/ Bluetooth-Scan . . . 3
1.1.4 /T80/ Broadcast-Test . . . 3
1.1.5 /T90/ Knotenanalyse und manuelle Steuerung per On-Screen-Joystick . . . 3
1.1.6 /T100/ Steuerung der Fahrtgeschwindigkeit . . . 3
1.1.7 /T110/ Steuerung per Beschleunigungssensor . . . 3
1.1.8 /T120/ Erkundungstest . . . 4
1.1.9 /T130/ erweiterter Steuerungstest . . . 4
1.1.10 /T140W/ Speichern der Kartendaten . . . 4
1.1.11 /T150W/ Laden der Kartendaten . . . 4
1.1.12 /T160W/ Test von globaler Lokalisierung . . . 4
1.1.13 /T170W/ Test der Zoomfunktionalität . . . 4
1.2 weitere Testfälle . . . 4

2 Unit Tests **4**
2.1 Android Anwendung . . . 4
2.1.1 GridMap . . . 4
2.1.2 Behaviour . . . 6
2.1.3 ComManager . . . 6
2.1.4 AStarPathFinder . . . 6
2.1.5 Handler . . . 6
2.2 E-puck Firmware . . . 7
2.2.1 Ringpuffer . . . 7

3 Blackbox Tests **7**

4 kontrollflussorientierte Testverfahren **7**

Zusammenfassung

Dieses Dokument soll einen Überblick über ...

1 Globale Testszenarien und Testfälle

1.1 Pflichtenheft Testfälle

1.1.1 /T50/ Kalibrierung

Nach ordnungsgemäßer Kalibrierung findet keine Fehlervisualisierung statt. Andernfalls beginnen die Außen LEDs zu blinken. (/F180/).

1.1.2 /T60/ Linienerkennung

Der e-puck wird im Betrieb entlang einer Linie aufgestellt und folgt dieser in Richtung der Liniensensoren (/F100/).

1.1.3 /T70/ Bluetooth-Scan

Mehrere e-puck-Roboter werden auf dem Spielfeld platziert und eingeschaltet. Nach der Initialisierungsphase sollen sämtliche Teilnehmer auf dem Smartphone erkannt werden (/F200/).

1.1.4 /T80/ Broadcast-Test

Gemäß /T70/ werden alle verbindungsbereiten e-puck Roboter auf dem Suchdialog des Smartphone angezeigt. Es wird ein Roboter ausgewählt, alle im Netzwerk befindlichen Teilnehmer werden im Dropdown-Steuerelement aufgelistet (/F50/, /F60/, /F65/, /F70/, /F205/, /F210/).

1.1.5 /T90/ Knotenanalyse und manuelle Steuerung per On-Screen-Joystick

Der Roboter wird gemäß /T60/ aufgestellt. Durch manuelle Steuerung per On-Screen-Joystick muss der Benutzer auf allen Knotentypen die möglichen Fahrtrichtungen testen. Nur gültige Richtungen dürfen vom Roboter befahren werden (/F80/, /F110/, /F220/, /F270/).

1.1.6 /T100/ Steuerung der Fahrtgeschwindigkeit

Der Roboter wird gemäß **/T60/** aufgestellt. Bei der manuellen Steuerung per On-Screen-Joystick werden sämtliche Geschwindigkeitsstufen überprüft (/F85/).

1.1.7 /T110/ Steuerung per Beschleunigungssensor

Der Test wird analog zu /T80/ durchgeführt, wobei als Steuerungsmethode der Beschleunigungssensor verwendet wird (/F280/).

1.1.8 /T120/ Erkundungstest

Mehrere e-puck Roboter werden auf entsprechende Startpositionen innerhalb des Spielfelds gesetzt und eingeschaltet. Ziel des Testszenarios ist die vollständige Erkundung durch effiziente Zusammenarbeit der Teilnehmer. Falls die Roboter nach Abschluss auf die Startpositionen zurückkehren und die Karte auf dem Smartphone dargestellt wird, ist der Testfall erfolgreich (/F90/, /F120/, /F130/, /F135/, /F140/, /F150/, /F155/, /F160/, /F190W/, /F230/, /F320/, /F330/).

1.1.9 /T130/ erweiterter Steuerungstest

Der Benutzer wählt nach der Lokalisierung von mindestens zwei Roboter einen per Dropdown-Steuerelement aus. Anschließend wird die Steuerung über beide Steuerungsarten (/F270/, /F280/) durchgeführt. Im nächsten Schritt wird eine anderer Roboter gewählt und die Steuerung mit diesem geprüft (/F290/, /F300/, /F310/).

1.1.10 /T140W/ Speichern der Kartendaten

Nachdem eine Kartenansicht auf dem Smartphone verfügbar ist, speichert der Benutzer die Kartendaten ab (/F250W/).

1.1.11 /T150W/ Laden der Kartendaten

Nachdem Kartendaten auf dem Smartphone gespeichert wurden (/T120W/), lädt der Benutzer diese. Daraufhin wird die entsprechende Karte angezeigt (/F260W/).

1.1.12 /T160W/ Test von globaler Lokalisierung

Die e-puck Roboter werden im Gegensatz zu /T110/ auf beliebigen Startpositionen gesetzt. Somit müssen sich die Teilnehmer zunächst finden und synchronisieren. Der erfolgreiche Abschluss kann durch die Anzeige aller e-pucks auf der Karte kontrolliert werden (/F170W/).

1.1.13 /T170W/ Test der Zoomfunktionalität

Nach dem Aufbau der Karte auf dem Smartphone werden Fingergesten zum Zoomen sowie Verschieben des Kartenausschnittes verwendet. (/F340W/).

1.2 weitere Testfälle

2 Unit Tests

2.1 Android Anwendung

2.1.1 GridMap

- `testInsertNode()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird die GridMap in eine Liste umgewandelt und deren Größe abgefragt, um sicherzustellen, dass der Knoten eingefügt wurde.

- `testFrontierNodeRightT()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob drei Frontierknoten eingefügt wurden.
- `testFrontierNodeLeftT()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob drei Frontierknoten eingefügt wurden.
- `testFrontierNodeTopT()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob drei Frontierknoten eingefügt wurden.
- `testFrontierNodeBottomT()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob drei Frontierknoten eingefügt wurden.
- `testFrontierNodeCross()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob vier Frontierknoten eingefügt wurden.
- `testFrontierNodeBottomRightEdge()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob zwei Frontierknoten eingefügt wurden.
- `testFrontierNodeBottomLeftEdge()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob zwei Frontierknoten eingefügt wurden.
- `testFrontierNodeTopRightEdge()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob zwei Frontierknoten eingefügt wurden.
- `testFrontierNodeTopLeftEdge()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird durch die GridMap eine Liste mit Frontierknoten erstellt und überprüft, ob zwei Frontierknoten eingefügt wurden.
- `testUpdateNode()` Es werden zwei neue Knoten erstellt und die GridMap eingefügt. Der zuletzt Eingefügte wird abgerufen und sein Knotentyp überprüft. Außerdem wird durch die GridMap erneut eine Liste mit Frontierknoten erstellt und deren Größe, die abhängig von der Wahl der Knotentypen ist, überprüft.
- `testMapBorders()` Es werden vier neue Knoten erstellt und in die GridMap eingefügt. Je nach Wahl der Koordinaten dieser Knoten wird hier

die Größe des Spielfeldes überprüft. Es werden vier Werte verglichen, der minimale x Wert, der maximale x Wert, der minimale y Wert und der maximale y Wert.

- `testSerialieMapInString()` Ein neuer Knoten wird erstellt und in die GridMap eingefügt. Anschließend wird die serialisiert und in einem String Array gespeichert. Dieses Array beinhaltet nun den x, den y Wert und den Knotentyp der zuvor eingefügten Knoten, in genau dieser Reihenfolge. Es wird überprüft, ob sich diese Werte entsprechen.

2.1.2 Behaviour

- `testExploreBehaviour()` Für den Test des Behaviours wurde eine vorgefertigte Map mit zwei Frontierknoten angelegt. Da diese Knoten bekannt sind, konnte so überprüft werden, ob durch das Behaviour der beste Knoten ausgewählt wurde. Dazu wurde eine Instanz des Interface IBehaviour angelegt und mit `execute()` die Liste der verschiedenen Behaviours durchlaufen. Als erstes entfernt das `removePathlessBehaviour()` unerreichbare Knoten, dann ermittelt das `DistanceBehaviour()` die Wegkosten von einem Startknoten zum Zielknoten, als nächstes werden durch das `InnerBehaviour()` unerkundete Knoten in bereits erkundeten Flächen weniger stark gewichtet, als Frontierknoten außerhalb dieser Flächen. Zuletzt wird durch das `CooperativeBehaviour()` festgestellt, dass sich nicht zwei Roboter auf ein und denselben Zielknoten zubewegen. Wenn diese Kette durchlaufen wird aus der Map der Knoten mit den geringsten Fahrtkosten ausgelesen und mit dem vorher bestimmten, besten Knoten verglichen.

2.1.3 ComManager

- `testAddClientAndSend()`
- `testRemoveClient()`

2.1.4 AStarPathFinder

- `testFindPuckMapNodeMapNodeArray()`

2.1.5 Handler

- `testSimTurnHandler()` Es wird eine neue Nachricht erzeugt, die aus 32 Byte besteht. Diese enthält einen Nachrichtenschlüssel, der wiederum aus den ersten zwei Byte besteht. Nun wird ein VirtualPuckRequest erzeugt, der an den Handler gesendet wird. Der Test ist bestanden, wenn sichn der Handler für seinen Nachrichtentyp zuständig fühlt und diese Nachricht bearbeitet.
- `testSimStatusHandler()`
- `testSimSpeedHandler()`
- `testSimResetHandler()`

- `testSimMoveHandler()`
- `testSimLEDHandler()`
- `testPuckStatusHandler()`
- `testPuckRejectHandler()`
- `testPuckOkHandler()`
- `testPuckNodeHitHandler()`
- `testPuckCollisionHandler()`
- `testPuckAbyssHandler()`
- `testFailureRequestHandler()`
- `testDriveRequestHandler()`
- `testControlledRequestHandler()`
- `testCollisionRequestHandler()`

2.2 E-puck Firmware

2.2.1 Ringpuffer

Auf dem Ringpuffer wurden die üblichen Funktionen, wie `pop()` und `push()` getestet. Zunächst wurde auf den leeren Puffer ein pop ausgeführt. Als Statusmeldung wurde -1 zurückgeliefert, was den Erwartungen entsprach `(Size: 32 Usage: 0 Free: 32)`. Dann wurden die Buchstaben a bis z gepusht `(IsEmpty: False Size: 32 Usage: 26 Free: 6)`. Anschließend wurden die Zahlen 0 bis 5 gepusht, um den Puffer zu füllen `(IsFull: True)`. Es wurde versucht ein weiteres Element zu pushen. Dies wurde vom Ringpuffer angelehnt und die Nachricht wurde verworfen. Zum Ende wurden alle Elemente gepopt `(Size: 32 Usage: 0 Free: 32)`.

3 Blackbox Tests

4 kontrollflussorientierte Testverfahren