1. Projektdaten (LOC, Packages, Klassen) (Metrics)
2. Testvoraussetzungen

* Wie haben wir getestet?
  + Android JUnit3

Nicht das aktuelle JUnit 4 weil es mit dem Android System nicht konform ist. Grund: Android nutzt eine eigene VM (Dalvik) die nicht mit JUnit 4 kompatibel ist. Hyperterminal / Putty

Es wird C Code auf den E-Puck aufgespielt und Statusmeldungen werden direkt auf der Konsole ausgelesen. Die Kommunikation funktioniert über Bluetooth.

* + GUI durch Benutzen

Anklicken aller Steuerelemente der Activities. Auf einen Test der Activities mittels JUnit wurde verzichtet, da der Aufwand in keinem Verhältnis zu der Erkenntnis steht. Z.B. Steuerung. Konnte nur real getestet werden.

1. Globale Testszenarien und Testfälle

Da das Projekt nicht mehr zu 100% dem Pflichtenheft entspricht, sind damit auch einige Testfälle unbrauchbar geworden.

* T100 Steuerung der Fahrtgeschwindigkeit

Auf die Kontrolle der Geschwindigkeit während der Erkundung und der manuellen Steuerung wurde komplett verzichtet.

* T160W Globale Lokalisierung

Auf eine globale Lokalisierung wurde ebenfalls verzichtet. Die Roboter kennen ihre eigene Startposition und die der anderen um eine erfolgreiche Erkundung durchzuführen. Umfang des Projekts. Aber Erweiterbarkeit durch Schichtenarchitektur! Leicht einzubauen.

* T170W Zoomfunktion der Karte

Die Autoskalierung der Karte steht dem Benutzer nicht mehr unmittelbar zur Verfügung. Während der Erkundung ist die Karte scrollbar, würde hier gezoomt werden wäre die Auswahl von Robotern aus der Karte heraus nicht mehr möglich. Die Autoskalierung findet jedoch beim Import sowie bei der Vorschau ihre Anwendung.

* Alle anderen Testfälle wurden durchgeführt und bestanden

Siehe Tabelle.

* Weitere Testfälle

Blackbox Tests der Activities:

Simulationskarte -> richtige Erkundung (Logik), richtiges Zeichen (SurfaceView)

Manuelle Steuerung (beide Methoden)

Statistik

Connect

Import / Export

1. Unit Tests

* Android:
* GridMap

testInsertNode() -> einfügen und anschließendes Auslesen

testFrontierNodeXXX() -> neuer Knoten erstellt, dieser erzeugt Frontierknoten

testUpdateNodes() -> ein Frontierknoten wird erkundet und erstellt wieder neue

testMapBorders() -> liefert maxX minX maxY minY

testSerializeMapToString() -> GridMap wird in Array umgewandelt und ausgelesen

* ComManager

testAddClientAndSend() -> hinzufügen von Client, Senden und Empfangen von Nachricht

testRemoveClient() -> hinzufügen und entfernen von Client, dann Nachricht senden

* Behaviour

testExploreBehaviour() -> mit vorgegebener Karte werden zwei Frontierknoten erstellt, dann wird überprüft ob verhalten den günstigsten Knoten auswählt. Removepathlessbehaviour(entfernt knoten die unerreichbar)

Distancebehaviour(A\* Gewichtung von Knoten)

Innerbehaviour(freie Felder innerhalb erkundeter Gebiete)

Cooperativebehaviour(knoten die bereits von anderen als ziel gewählt wurden sind teurer)

* AStarPathFinder

testFindPuckMapNodeMapNodeArray() -> Prüftrüft ob kürzester von einem Startknoten zu 3 Zielkonoten gefunden wird. Pfad wird geprüft.

* Handler

testXXXXXXHandler() -> prüft ob sich Handler für diese Nachricht verantwortlich fühlt

- EPuck:

- Ringpuffer

Alle wichtigen funktionen, einfügen, entfernen von einzelnen elementen und von mehrere elementen gleichzeitig.

1. Herausforderungen und Probleme des Projekts

* Knotenerkennung des e-puck Roboters

Weiße Flächen auf den Knotenpunkten erforderlich (Regler wirkt entgegen) Sensor erkennt Knoten -> Regler wird abgeschalten

* Kollisionserkennung der Roboter auf dem Spielfeld

Müssen auf vorherigen Knoten zurückfinden. Vor oder hinter knoten? Schwellwert -> mit Schrittzähler zurückfahren ohne Regler. Andernfalls mit Regler.

* RaceConditions

Synchronisierung von Zustandsdaten der PuckInstanzen. LogicThread und Bluetooth/BroadcastCommunication greifen auf PuckInstanz zu. Jeweils nur einer kann darauf zugreifen. Variablen werden z.b. nur zur Hälfte verändert -> falsches Verhalten