Robotik - Übung 01

Grundlagen der Roboterprogrammierung

Christoph Zinnen

WS17/18

LABORROBOTIK
Informatik - Computer Science

Informatik
Hauptcampus

HOCHSCHULE TRIER

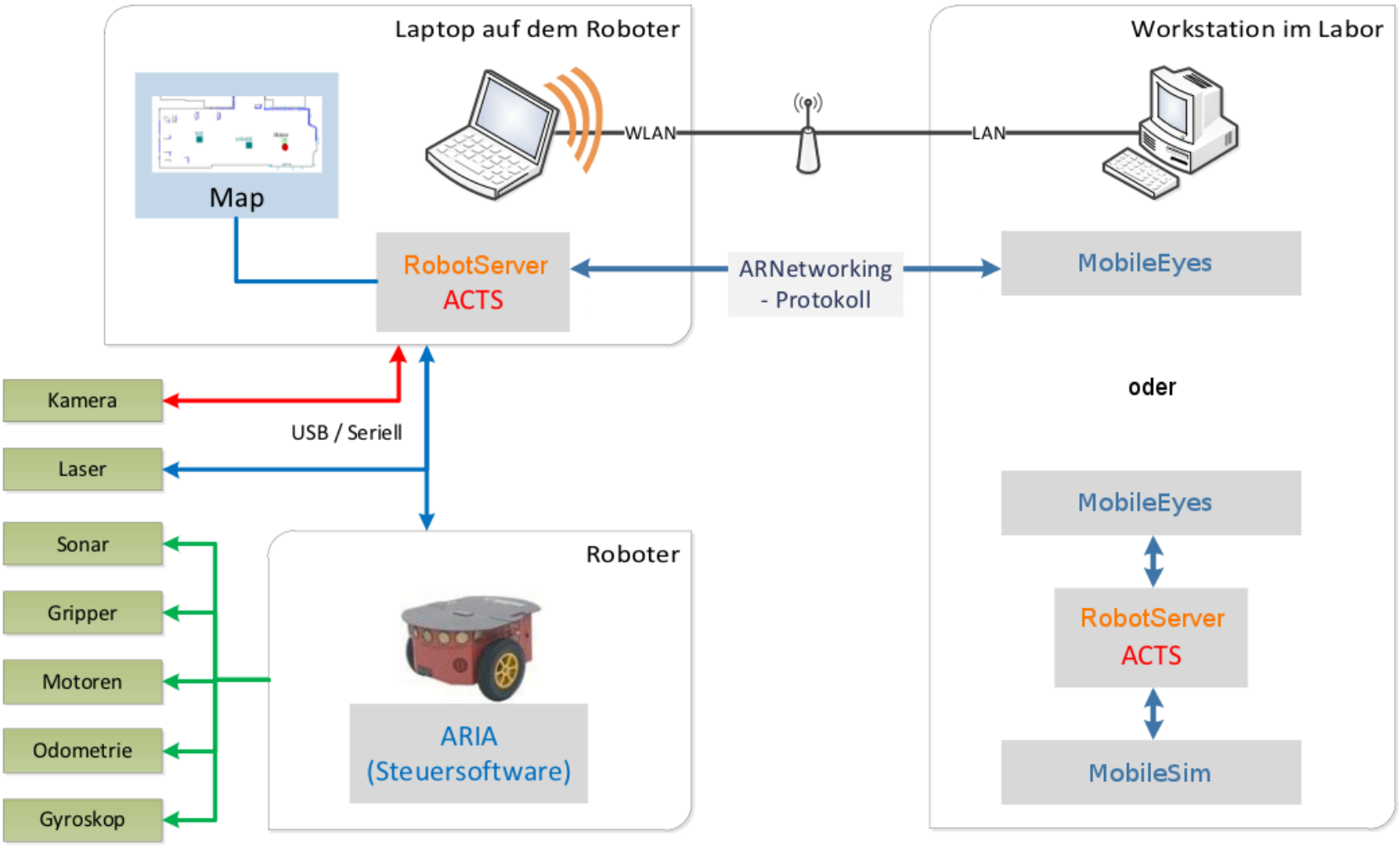
Gliederung

Gliederung

Entwicklungsumgebung

- Betriebssystem: **Linux**
- Versionsverwaltung: **Git**
- Programmiersprache: **C++**
- Build System: **CMake**
- Entwicklungsumgebung: **QtCreator**
- Roboter Simulator: **MobileSim**
- Grafische Benutzeroberfläche: **MobileEyes**
- Objekterkennung: **ACTS** (**A**dvanced **C**olor **T**racking **S**ystem)

Software Architektur



Motivation

Warum C++?

- Breiter Einsatz in der Industrie(Im Bereich Robotik)!
- Primäre Sprache für die eingesetzten Systeme.

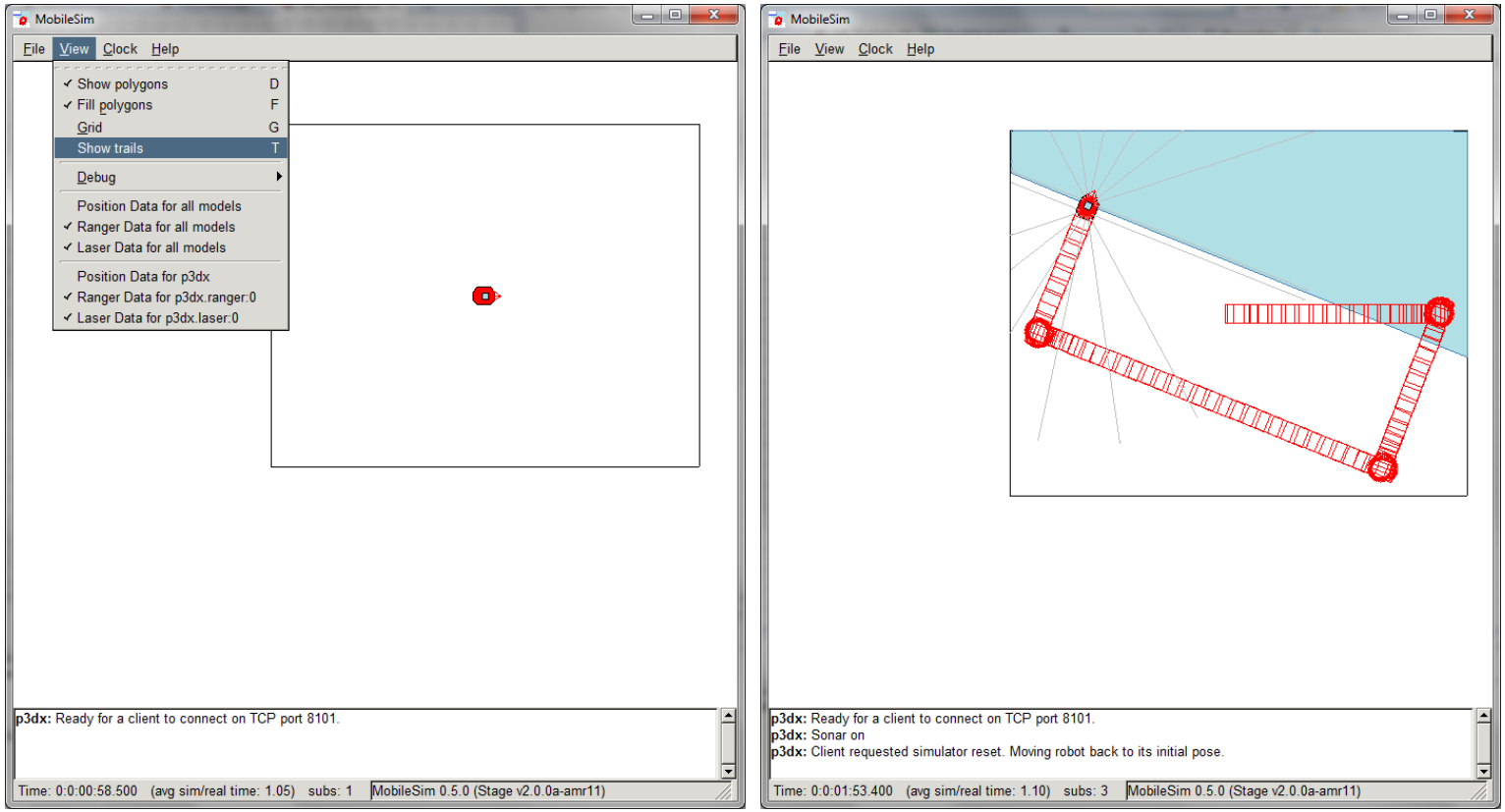
Warum nicht mehr Python?

- Schlechte Unterstüzung durch die eingesetzten Systeme.
- Wrapper Schicht sorgt für zusätzliche Fehler.

Warum nicht Java?

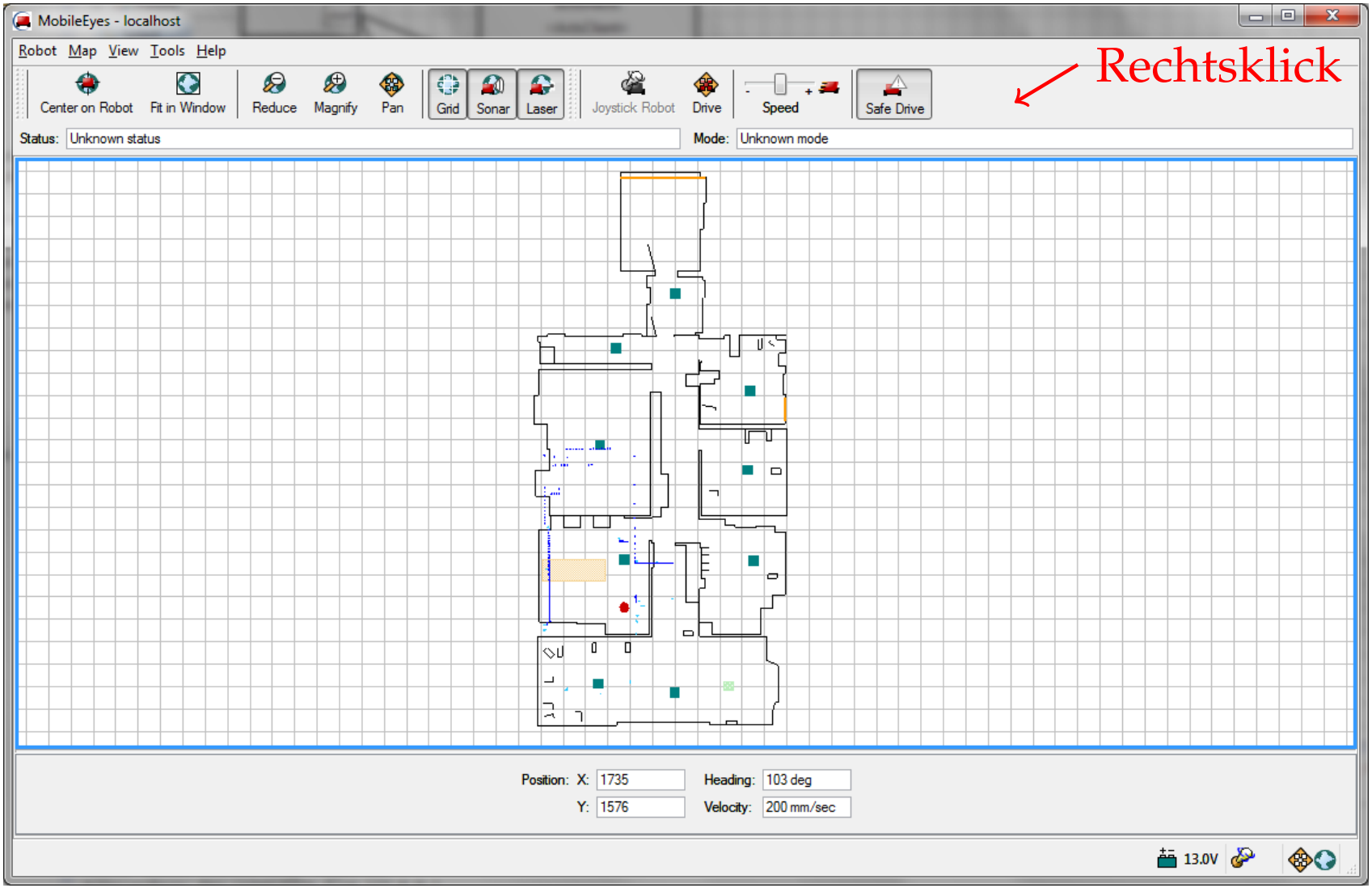
- Geringe Relevanz in der Industrie(Im Bereich Robotik).
- Schlechte Unterstützung durch existierende Systeme.

MobileSim

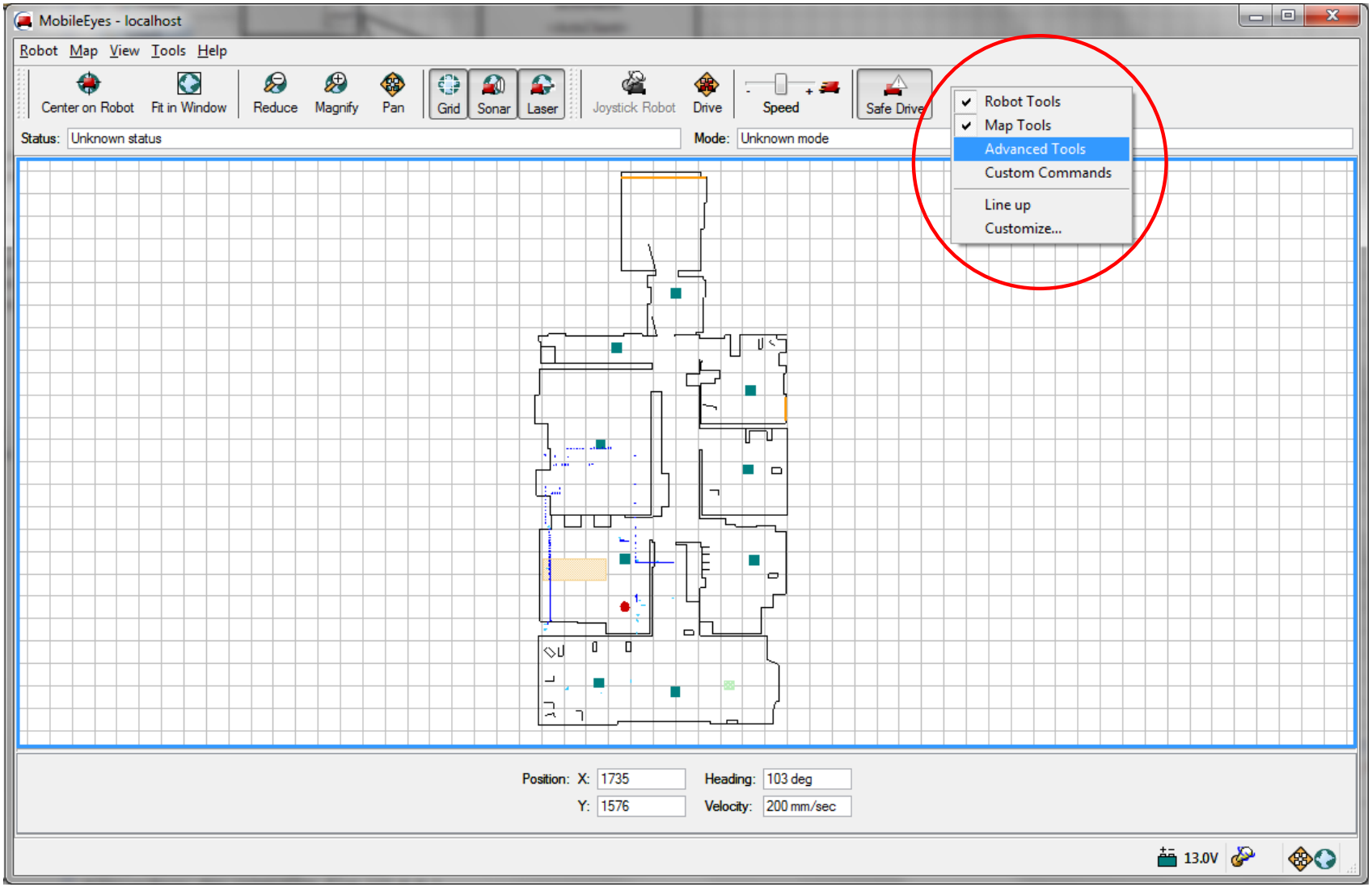


Die Option *Show trails* [T] zeichnet in regelmäßigen Abständen ein rotes Rechteck an die Position des Roboters. Damit lässt sich der vom Roboter zurückgelegte weg nachvollziehen.

MobileEyes

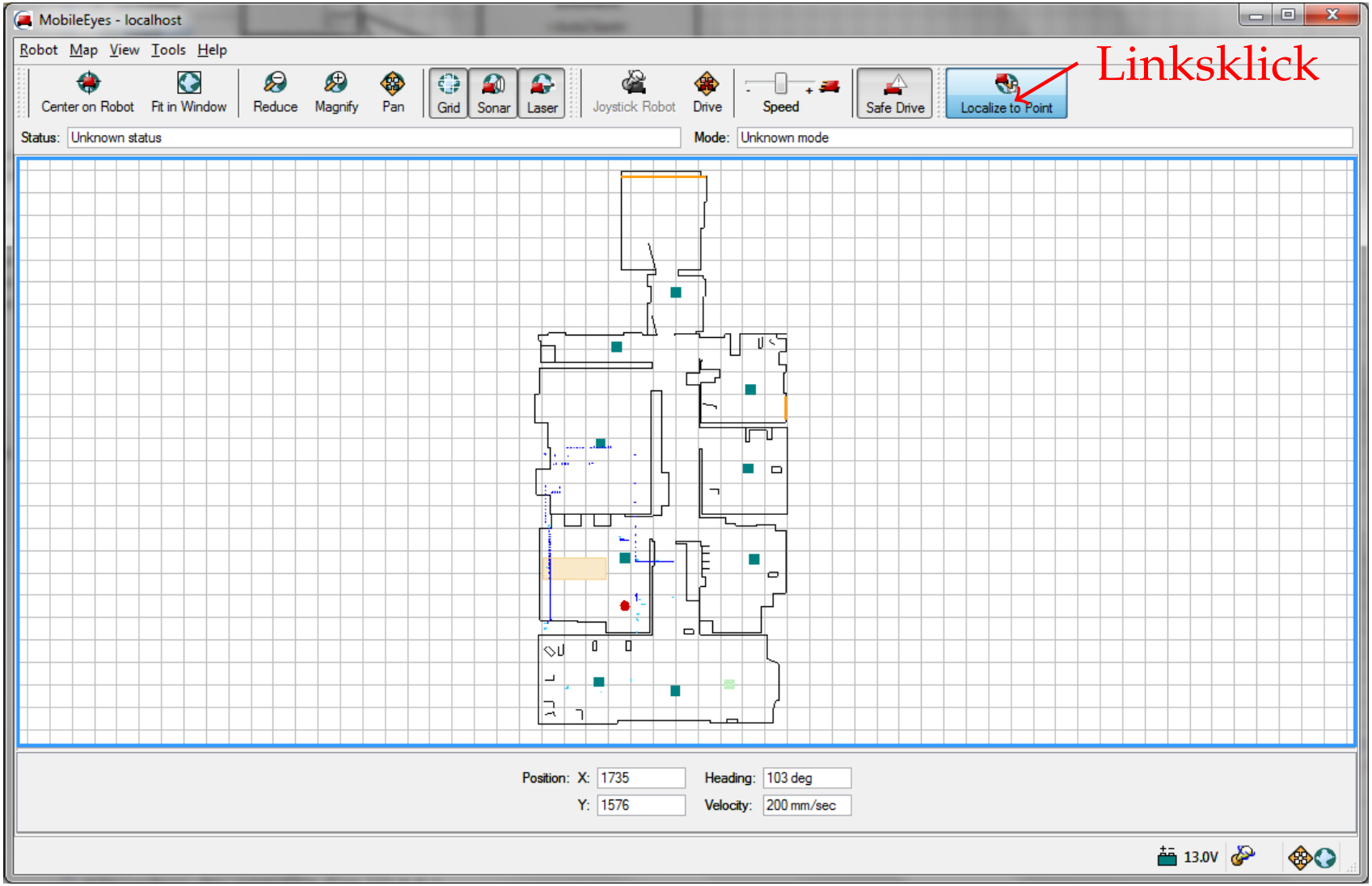

MobileEyes - localhost
Robot Map View Tools Help
Center on Robot
Fit in Window
Reduce
Magnify
Pan
Grid
Sonar
Laser
Joystick Robot
Drive
Speed
Safe Drive
Rechtsklick
Status: Unknown status
Mode: Unknown mode
Position: X: 1735
Y: 1576
Heading: 103 deg
Velocity: 200 mm/sec
13.0V

MobileEyes

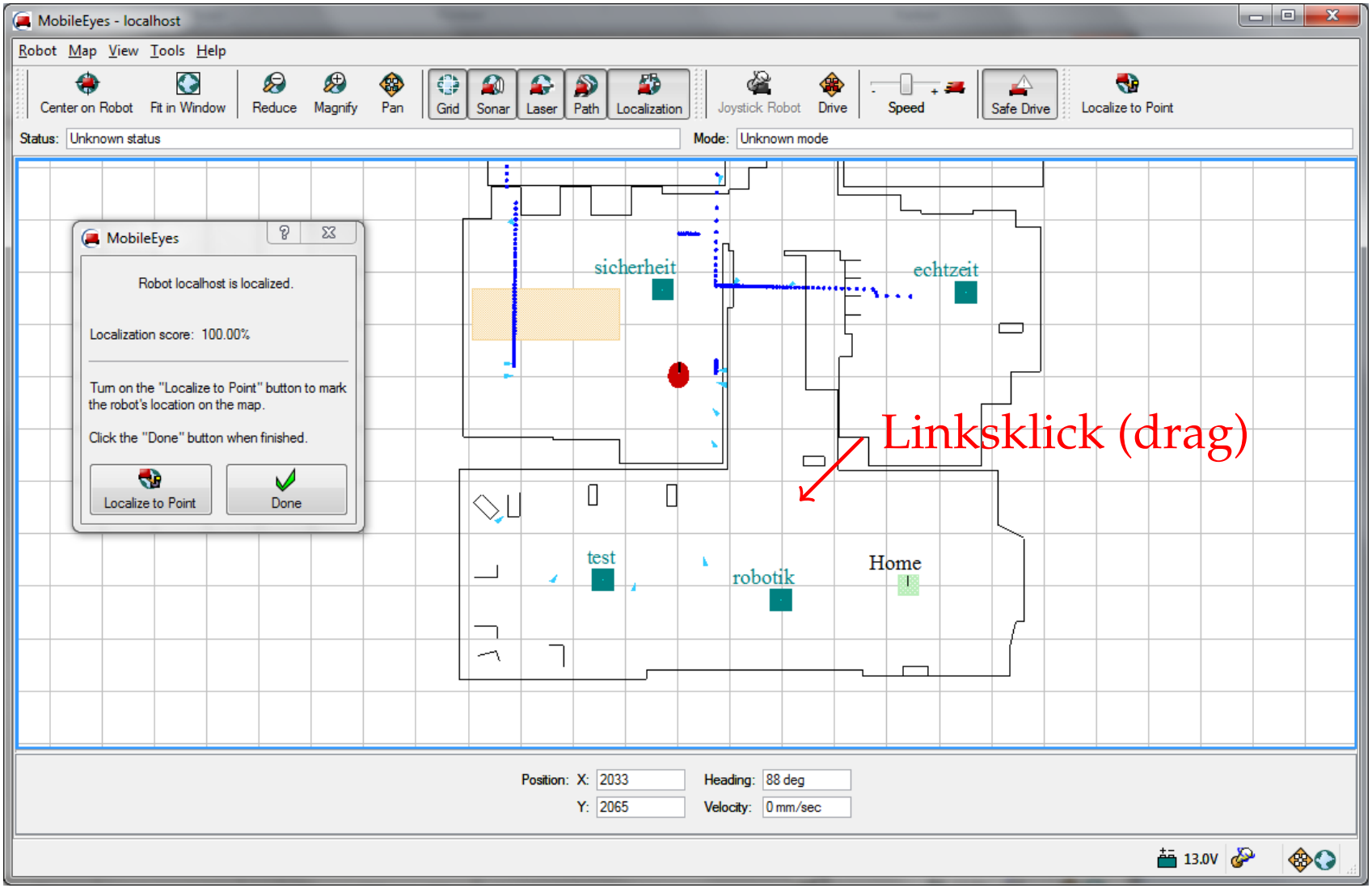


Hinzufügen des Werkzeugs zur manuellen Lokalisierung.

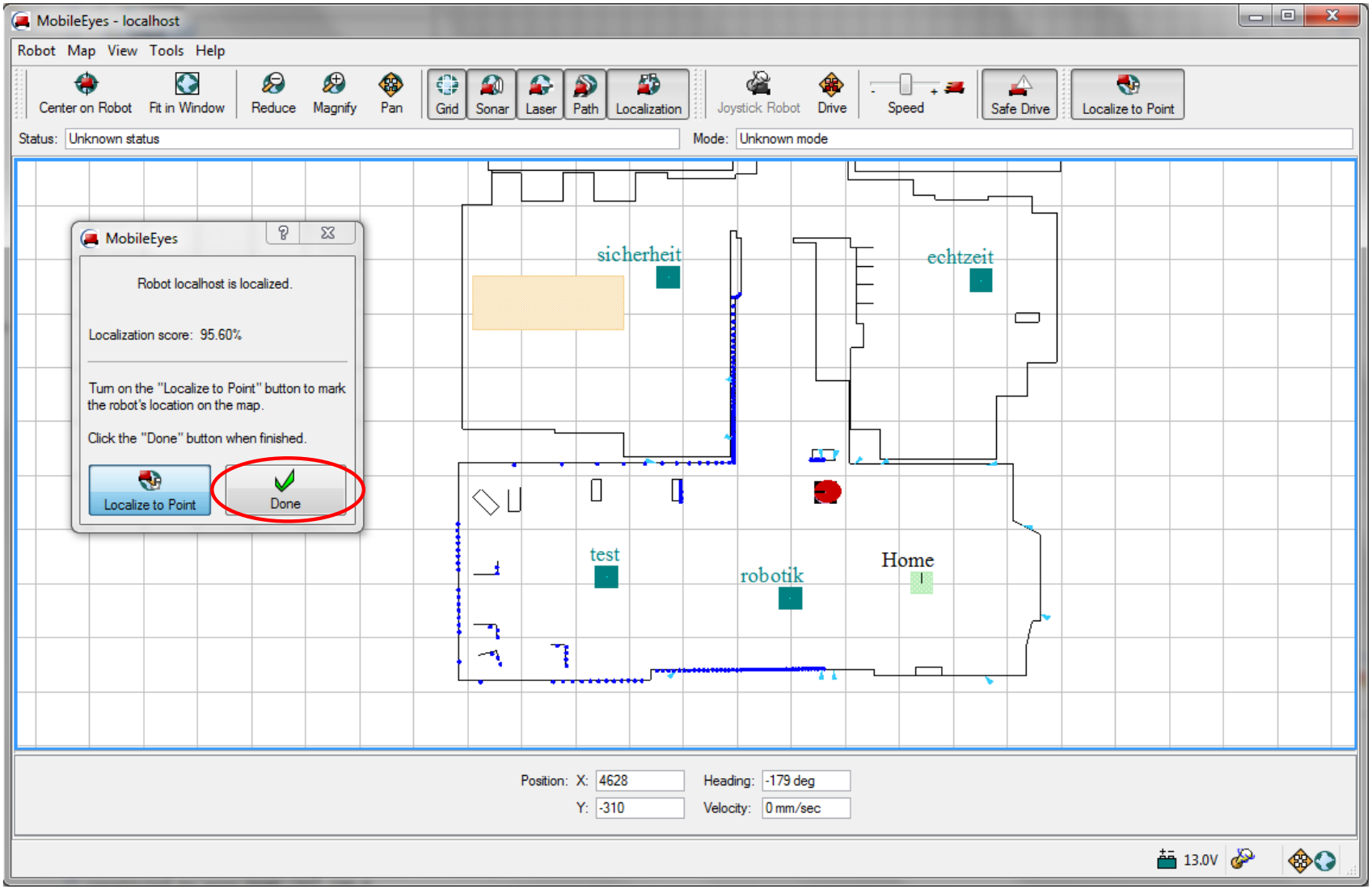
MobileEyes


MobileEyes - localhost
Robot Map View Tools Help
Center on Robot
Fit in Window
Reduce
Magnify
Pan
Grid
Sonar
Laser
Joystick Robot
Drive
Speed
Safe Drive
Localize to Point
Linksklick
Status: Unknown status
Mode: Unknown mode
Position: X: 1735
Y: 1576
Heading: 103 deg
Velocity: 200 mm/sec
13.0V

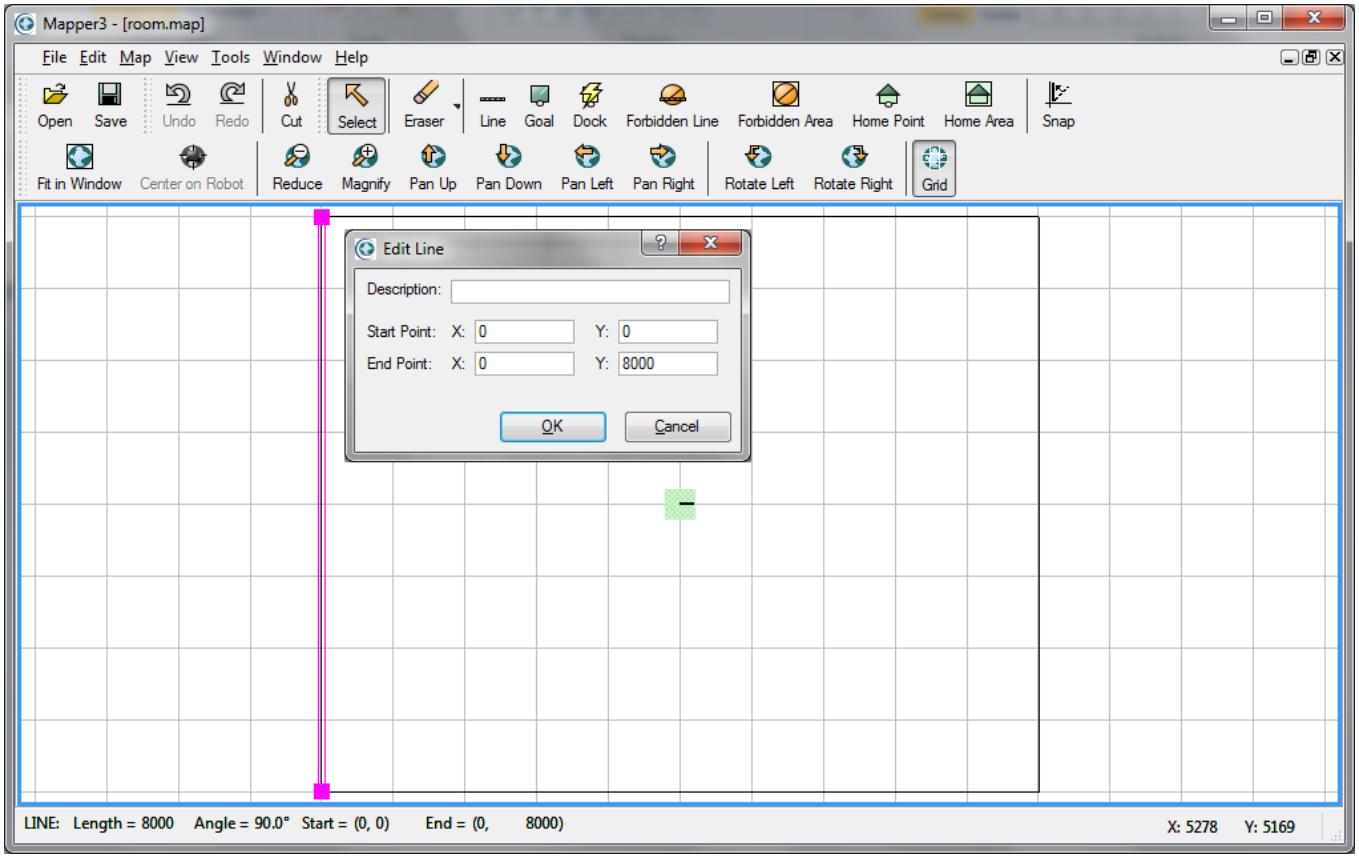
MobileEyes


MobileEyes - localhost
Robot Map View Tools Help
Center on Robot
Fit in Window
Reduce
Magnify
Pan
Grid
Sonar
Laser
Path
Localization
Joystick Robot
Drive
Speed
Safe Drive
Localize to Point
Status: Unknown status
Mode: Unknown mode
MobileEyes
Robot localhost is localized.
Localization score: 100.00%
Turn on the "Localize to Point" button to mark the robot's location on the map.
Click the "Done" button when finished.
Localize to Point
Done
sicherheit
echtzeit
Linksklick (drag)
test
robotik
Home
Position: X: 2033
Y: 2065
Heading: 88 deg
Velocity: 0 mm/sec
13.0V

MobileEyes


MobileEyes - localhost
Robot Map View Tools Help
Center on Robot
Fit in Window
Reduce
Magnify
Pan
Grid
Sonar
Laser
Path
Localization
Joystick Robot
Drive
Speed
Safe Drive
Localize to Point
Status: Unknown status
Mode: Unknown mode
MobileEyes
Robot localhost is localized.
Localization score: 95.60%
Turn on the "Localize to Point" button to mark the robot's location on the map.
Click the "Done" button when finished.
Localize to Point
Done
sicherheit
echtzeit
test
robotik
Home
Position: X: 4628
Y: -310
Heading: -179 deg
Velocity: 0 mm/sec
13.0V

Mapper3



Der *Edit Line* Dialog öffnet sich durch einen Doppelklick auf eine Linie.

Gliederung

Hello World Action I

(Default)RobotServer (Eigenentwicklung)
Kümmert sich um die Initialisierung des Roboters und stellt Basiskomponenten(Sonar, Laser, Map, uvm.) bereit.

```
1  #include <defaultrobotserver.h>
2  #include "helloWorldAction.h"
3
4  int main(int argc, char **argv) {
5      DefaultRobotServer server;
6      server.init(argc, argv);
7
8      HelloWorldAction helloWorldAction;
9      server.addAction(helloWorldAction);
10     helloWorldAction.activate();
11
12     server.run();
13     return 0;
14 }
```

Listing 1: main.cpp

Hello World Action II

ArAction
Basisklasse für Verhaltenskomponenten (Behaviours).

```
1  #include <Aria.h>
2
3  class HelloWorldAction : public ArAction
4  {
5  public:
6      HelloWorldAction();
7      ArActionDesired *fire(ArActionDesired currentDesired);
8  };
```

Listing 2: helloworldaction.h

Konstruktor und `fire` Methode müssen für jede Action implementiert werden.

Hello World Action III

```
1  #include "helloWorldAction.h"
2
3  HelloWorldAction::HelloWorldAction() :
4      ArAction("HelloWorldAction",
5               "This action just prints \"Hello World\"."){
6  }
7
8  ArActionDesired *HelloWorldAction::fire(ArActionDesired currentDesired)
9  {
10     ArLog::log(ArLog::Normal, "Hello World!");
11     this->deactivate();
12     return 0;
13 }
```

Listing 3: helloworldaction.cpp

Im Konstruktor muss der Konstruktor von `ArAction` mit einem Namen und einer Beschreibung der Action aufrufen. Die Methoden `activate()` und `deaktivate()` können zum starten bzw. stoppen einer Action verwendet werden.

Roboter-Informationen I

Ist der Winkel θ der Pose des Roboters 0°, so ist er nach rechts (Osten) bezüglich seiner Karte ausgerichtet.
Positive Winkel kleiner 180° bewirken eine Drehung nach links, negative größer -180° nach rechts.

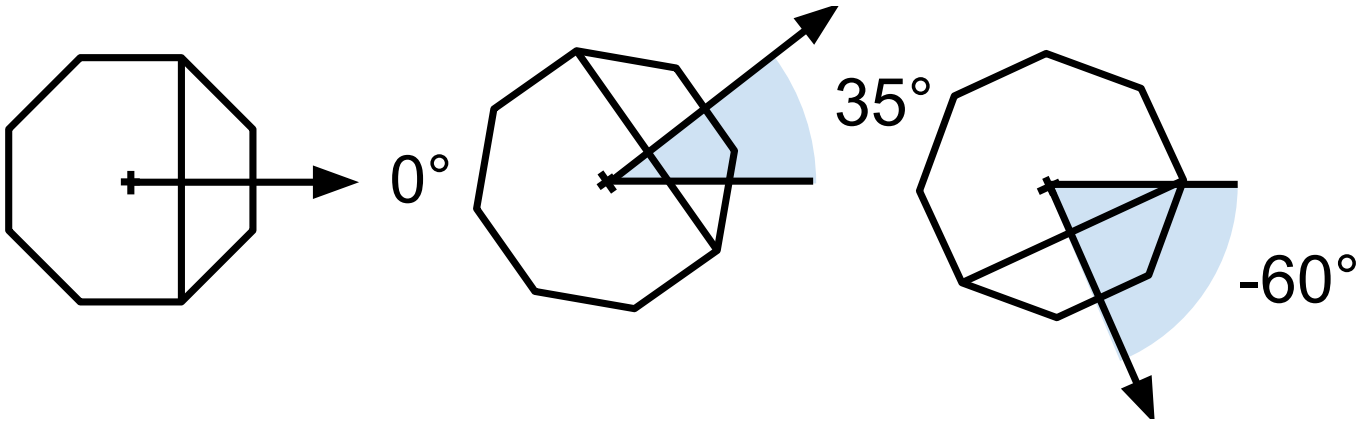


Abbildung 1: Roboterausrichtung

Roboter-Informationen II

ArRobot: Schnittstelle zum Roboter(Pose, Radius, Sonar-Senoren, Motoren, usw.).

```
10  ArActionDesired *RobotInfoAction::fire(ArActionDesired currentDesired)
11  {
12      double x,y,th, radius;
13
14      x = myRobot->getX();
15      y = myRobot->getY();
16      th = myRobot->getTh();
17      radius = myRobot->getRobotRadius();
18
19      ArLog::log(ArLog::Normal,
20                 "Pose: (%5.0fmm, %5.0fmm, %3.2f°), Radius: %5.0f)",
21                 x, y, th, radius);
22
23      return 0;
24  }
```

Listing 4: robotinfoaction.cpp

DirectAction I

- Direkte Steuerung des Roboters.
- Die Steuerungsbefehle werden auf dem `ArRobot` Objekt aufgerufen und direkt ausgeführt.
- Ein Befehl überschreibt evtl. aktiven Befehl.
- Ungeeignet für parallele Ausführung.

DirectAction II

- `void ArRobot::move(double distance`[*mm*])
  Fahre die übergebene Strecke.
- `void ArRobot::setVel(double velocity`[*mm*/*s*])
  Fahre mit der übergebenen Geschwindigkeit.
- `void ArRobot::setDeltaHeading(double heading`[°])
  Rotiere um den übergebenen Winkel(relativ zum Roboter).
- `void ArRobot::setHeading(double heading`[°])
  Rotiere zum übergebenen Winkel(relativ zur Karte).
- `void ArRobot::setRotVel(double velocity`[°/*s*])
  Rotiere mit der übergebenen Geschwindigkeit.

Behaviour

- Indirekte Steuerung des Roboters.
- Die Steuerungsbefehle werden auf einem `ArActionDesired` Objekt aufgerufen und in diesem gespeichert. Sie stellen Wünsche an den Roboter dar.
- Ein Pointer auf das Objekt muss am Ende der `fire` Methode an das Roboter System zurückgegeben werden.
- Die `ArActionDesired` Objekte aller Behaviors werden vom Roboter System ausgewertet und gemeinsam ausgeführt.

Durch die Verwendung von DirectActions werden Behaviours deaktiviert. Sie können mit `ArRobot::clearDirectMotion()` aktiviert werden.

Sonarsensoren I

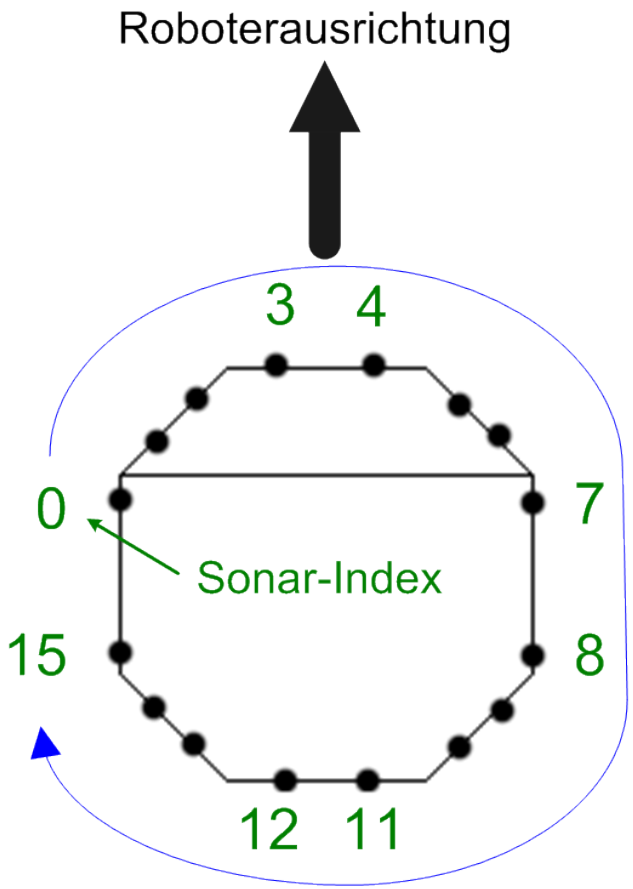


Abbildung 2: Anordnung der Sonarsensoren

Sonarsensoren II

- Die Anzahl der Sonarsensoren kann mit `ArRobot::getNumSonar()` ermittelt werden.
- Mit `int ArRobot::getSonarRange(int i)` kann die Entfernung(in *mm*) zum nächsten Hindernis eines Sensors ermittelt werden.
- Ist kein Hindernis in Reichweite, wird die maximale Entfernung von 5000*mm* zurückgegeben.

Sonarsensoren III

```
11  ArActionDesired *SonarRangeAction::fire(ArActionDesired currentDesired)
12  {
13      if (!myRobot) {
14          ArLog::log(ArLog::Terse, "Error: SonarRangeAction requires a robot!");
15          deactivate();
16          return 0;
17      }
18
19      int distance;
20      for (int i = 0; i < myRobot->getNumSonar(); ++i) {
21          distance = myRobot->getSonarRange(i);
22          ArLog::log(ArLog::Normal, "Sonar: %2d, Distance: %5d", i, distance);
23      }
24
25      return 0;
26  }
```

Listing 5: sonarrangeaction.h

Sonarsensoren IV

- `ArSensorReading *ArRobot::getSonarReading(int i)` liefert ein Reading eines Sensors.
- Ein Reading enthält die Entfernung(`ArSensorReading::getRange()`) und Pose(`ArSensorReading::getPose()`) des Hindernisses.
- Die Pose des Readings ist relativ zum Roboter.
- Ist kein Hindernis in Reichweite, wird die maximale Entfernung von 5000*mm* zurückgegeben.

Sonarsensoren V

```
11  ArActionDesired *SonarReadingAction::fire(ArActionDesired currentDesired)
12  {
13      if (!myRobot) {
14          ArLog::log(ArLog::Normal,
15                     "Error: SonarReadingAction requires a robot!");
16          deactivate();
17          return 0;
18      }
19
20      for (int i = 0; i < myRobot->getNumSonar(); ++i) {
21          ArSensorReading *reading = myRobot->getSonarReading(i);
22          if (reading) {
23              ArPose pose = reading->getPose();
24              ArLog::log(ArLog::Normal,
25                         "Sonar: %2d, Reading: (%5.0f, %5.0f), Distance: %5d",
26                         i, pose.getX(), pose.getY(),
27                         reading->getRange());
28          }
29      }
30      return 0;
31  }
```

Listing 6: sonarreadingaction.h

Sonar-/LaserReadingBox

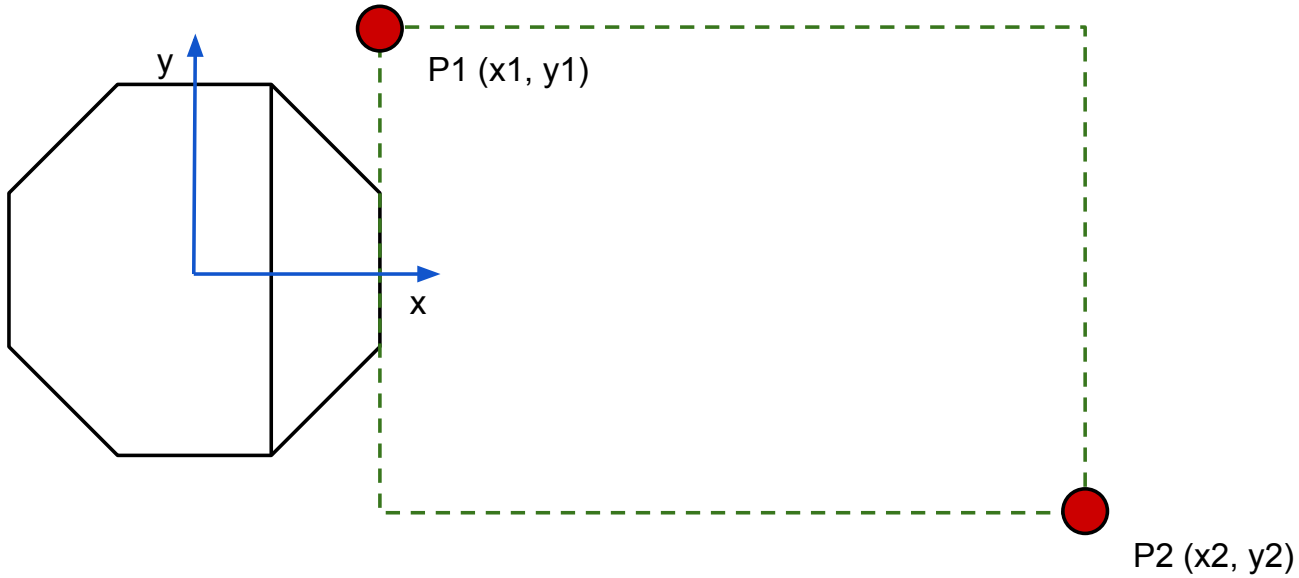


Abbildung 3: Eine Sensor Box vor dem Roboter

Sensor Boxen werden im Koordinatensystem des Roboters angegeben. Der Nullpunkt befindet sich dabei im Zentrum, die X-Achse verläuft in Fahrtrichtung, und die Y-Achse nach links.

SonarReadingBox I

```
8   class SonarReadingBoxAction : public ArAction
9   {
10  public:
11      SonarReadingBoxAction(ArSonarDevice *sonar);
12      ArActionDesired *fire(ArActionDesired currentDesired);
13
14  protected:
15      ArSonarDevice *sonar;
16      ArPose readingPose;
17      double x1, y1, x2, y2;
18
19      void init();
20      bool initialized;
21  };
```

Listing 7: sonarreadingboxaction.h

SonarReadingBox II

```
38  void SonarReadingBoxAction::init()
39  {
40      double robotRadius = myRobot->getRobotRadius();
41      double boxLength = 5000;
42      double boxWidth = 2 * (robotRadius + 100);
43
44      x1 = robotRadius;
45      y1 = -boxWidth * 0.5;
46      x2 = x1 + boxLength;
47      y2 = y1 + boxWidth;
48
49      initialized = true;
50  }
```

Listing 8: sonarreadingboxaction.cpp(1)

SonarReadingBox III

```
13  ArActionDesired *SonarReadingBoxAction::fire(ArActionDesired currentDesired)
14  {

    ...
22      if (!initialized) {
23          init();
24      }
25      unsigned int maxRange = sonar->getMaxRange();
26      double distance = sonar->currentReadingBox(x1, y1, x2, y2, &readingPose);
27
28      ArLog::log(ArLog::Normal, "SonarReadingBox: (%5.0f, %5.0f, %5.0f, %5.0f)",
29                 x1, y1, x2, y2);
30      ArLog::log(ArLog::Normal,
31                 "Position: (%5.0f, %5.0f), Distance: %5.0f, Range: %5.0d",
32                 readingPose.getX(), readingPose.getY(),
33                 distance, maxRange);
34
35      return 0;
36  }
```

Listing 9: sonarreadingboxaction.cpp(2)

LaserReadingBox I

```
6   class LaserReadingBoxAction : public ArAction
7   {
8   public:
9       LaserReadingBoxAction(ArLaser *laser);
10      ArActionDesired *fire(ArActionDesired currentDesired);
11
12      virtual void activate();
13
14  protected:
15      ArLaser *laser;
16      ArPose reading;
17      double x1, x2, y1, y2;
18
19      void init();
20      bool initialized;
21  };
```

Listing 10: laserreadingboxaction.h

LaserReadingBox II

```
14  void LaserReadingBoxAction::activate()
15  {
16      ArLog::log(ArLog::Normal, "LaserReadingBox activate()!");
17      if (!myRobot) {
18          ArLog::log(ArLog::Terse,
19                     "Error: LaserReadingBoxAction requires a robot!");
20          deactivate();
21      } else if (!laser) {
22          ArLog::log(ArLog::Terse,
23                     "Error: LaserReadingBoxAction requires a laser!");
24          deactivate();
25      } else {
26          init();
27
28          ArAction::activate();
29      }
30  }
```

Listing 11: laserreadingboxaction.cpp(1)

LaserReadingBox III

```
32  void LaserReadingBoxAction::init()
33  {
34      double robotRadius = myRobot->getRobotRadius();
35      double boxLength = 10000;
36      double boxWidth = 2 * (robotRadius + 100);
37
38      x1 = robotRadius;
39      y1 = -boxWidth * 0.5;
40      x2 = x1 + boxLength;
41      y2 = y1 + boxWidth;
42
43      initialized = true;
44  }
```

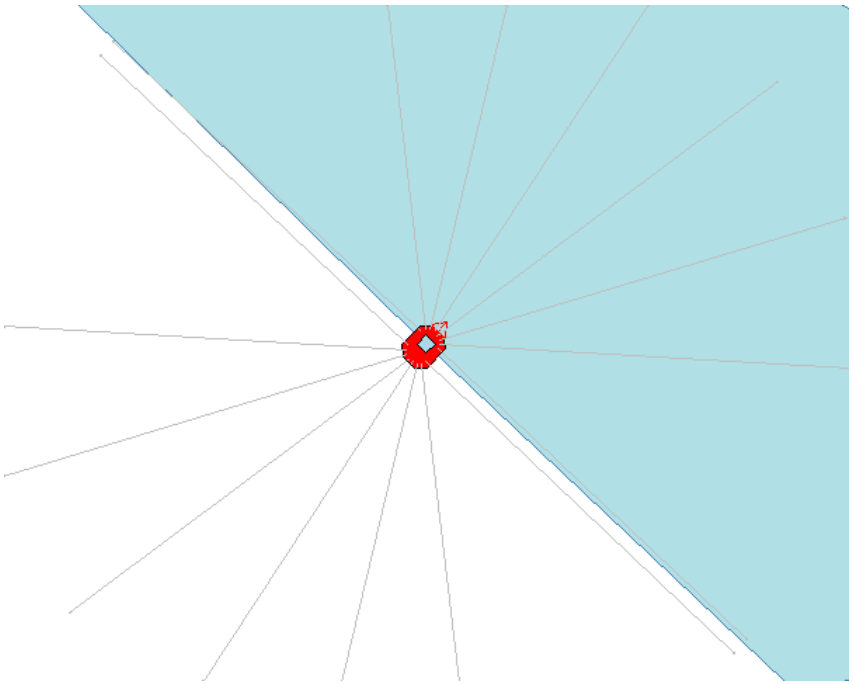
Listing 12: laserreadingboxaction.cpp(2)

LaserReadingBox IV

```
46  ArActionDesired *LaserReadingBoxAction::fire(ArActionDesired currentDesired)
47  {
48      if(!initialized) {
49          init();
50      }
51
52      laser->lockDevice();
53
54      if (laser->isConnected()) {
55          unsigned int maxRange = laser->getMaxRange();
56          double distance = laser->currentReadingBox(x1, y1, x2, y2,
57                                                     &reading);
58          ArLog::log(ArLog::Normal,
59                     "LaserReadingBox: (%5.0f, %5.0f, %5.0f, %5.0f)",
60                     x1,y1,x2,y2);
61          ArLog::log(ArLog::Normal,
62                     "Pos: (%5.0f, %5.0f), Dist: %5.0f, Range: %5.0d",
63                     reading.getX(), reading.getY(),
64                     distance, maxRange);
65
66      } else if (laser->isTryingToConnect()) {
67          ArLog::log(ArLog::Normal, "Connecting to Laser %s, please wait.",
68                     laser->getName());
69      } else {
70          laser->asyncConnect();
71      }
72
73      laser->unlockDevice();
74      return 0;
75  }
```

Listing 13: laserreadingboxaction.cpp(3)

LaserReadingBox V



- Gegenüber dem Sonar bietet der Laser eine höhere genauigkeit sowie verlässlichkeit.
- Er hat einen Sichtbereich von 180° (von -90° bis 90°).

Gliederung

1.Aufgabe: Verständnis

1. Schauen Sie sich die auf den Folien vorgestellten Code-Beispiele nochmal genau an und versuchen Sie diese zu Verstehen.
2. Falls Ihnen etwas unklar sein sollte, zögern Sie bitte nicht Fragen zu stellen.

Hinweis:

Am Privat-PC kann eine VM [1] mit der vorinstallierter Software eingesetzt werden

[1] `https://alfresco.hochschule-trier.de/share/proxy/alfresco-noauth/api/internal/shared/node/RtKXZKxwRBOntvSThjsf7A/content/Robotik-VM-14-04.zip`

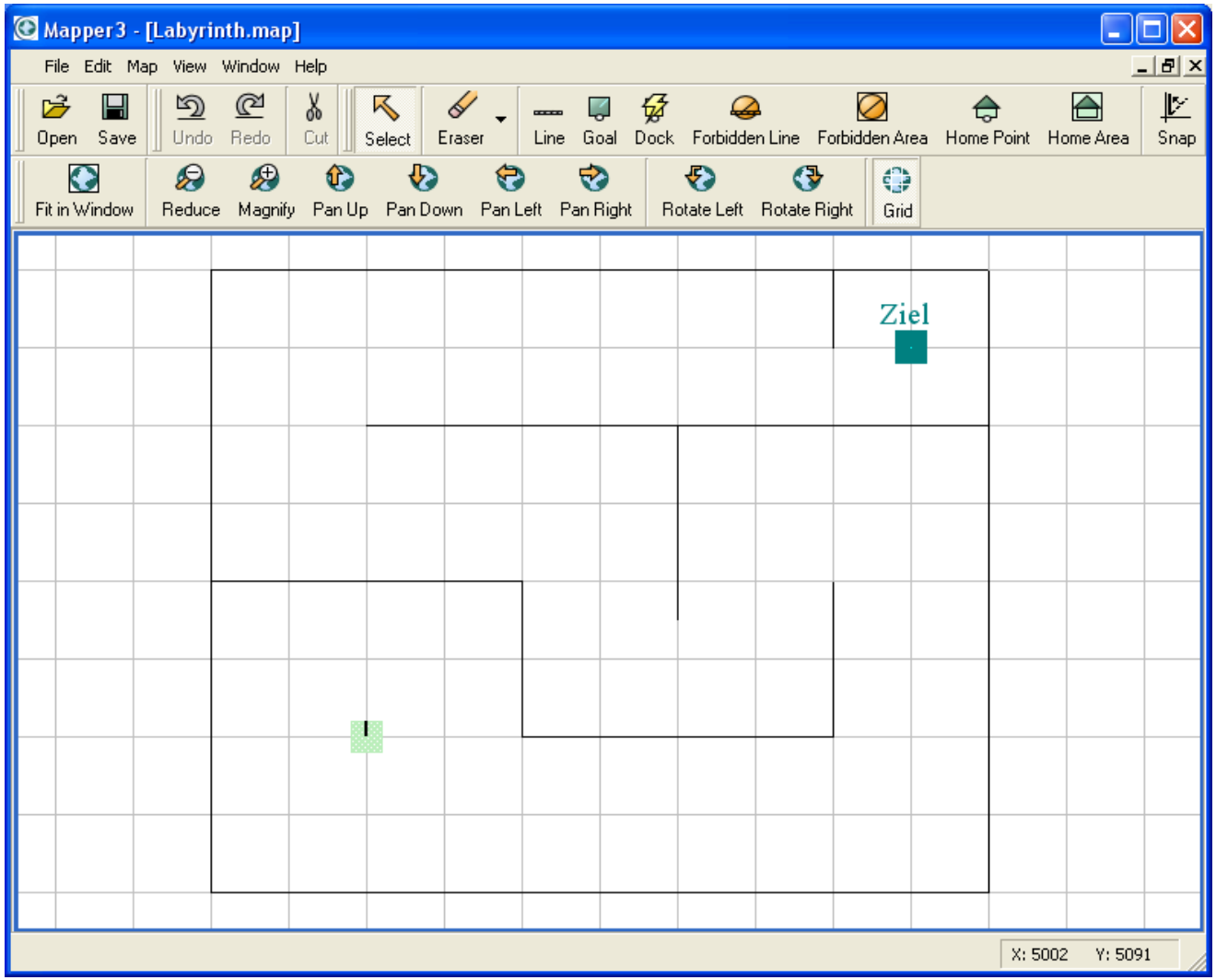
2. Aufgabe: Aktorik

1. Entwickeln Sie eine Action, die mithilfe von **Direct Action Commands** vom Startpunkt zum Ziel fährt.
2. Planen Sie zunächst die Fahrtroute (Digital oder auf Papier).
3. Starten Sie die von Ihnen entwickelte Action und beobachten Sie das Verhalten des Roboters.

Hinweis:

In dieser Aufgabe sollen keine Sensoren verwendet werden!

2. Aufgabe (Fortsetzung)



3. Aufgabe: Sensorik

Verbessern Sie Ihre Lösung aus Aufgabe 2. Verwenden Sie dazu die Sensoren.