Localization

Mapping

Lecture 5

October 11th 2016

Plan for today

- Calibrating the chassis for accurate poses.
- Alternative pose providers
- Localization, Mapping

  Practice it!

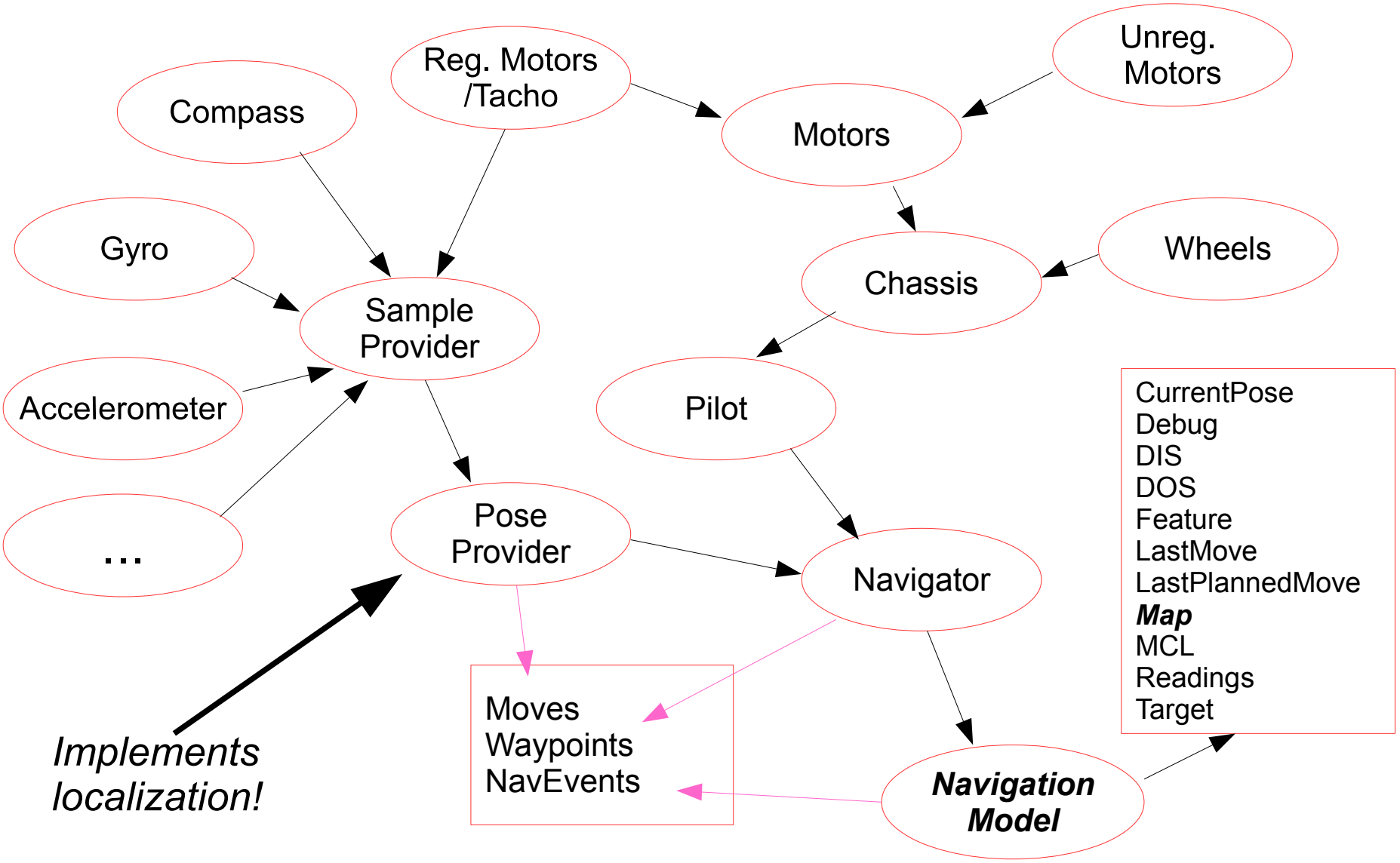
- *Feature detection*
- *Pathfinding*

  Practice it!

Navigation revisited

- Motor – control motors directly
- Pilot – use robot dimensions to calculate movements
- Navigator – calculate movements to reach a point in a coordinate system
- PoseProvider – keep track of position – dead-reckoning
- … do not mix levels in a program

Navigation revisited



Pose Providers

Pose provider is a class that determines the robots *Pose.*

All pose providers must implement the *pose provider interface (setPose(Pose), getPose().*

Pose(float x, float y, float heading)

Standard pose-provider use tachometer-odometry and dead-reckoning.

Other pose providers are possible, LejOS includes also classes:

BeaconPoseProvider

CompassPoseProvider

MCLPoseProvider

Different Communication *models*

1. Lego robot as stand-alone program
2. Client-server techniques: run program on EV3 and send output to console on (PC as last week)
3. *Use dataStreams and threads between program on EV3 and program on PC*
4. *Navigation Models support Client-Server configuration*

EV3NavigationModel

- is a way to integrate all the leJOS navigation features, and communicate with an application on the PC.
- The PC application typically shows your robot moving around in a mapped area, and lets you control the robot using all the leJOS navigation features.
- For example, you can use path finders to calculate paths, show them on map, and then follow them. While the robot moves around, any features or obstacles detected can be shown on the map.

- We will try out MapCommand implementation today

Localization

- Odometry, dead-reckoning, etc..
- Use the the kinematic model of differential drive to compute an expected position.

Problems:

- Friction, backlash, wrong measurements of wheels, variation of contact points

Calibrate!

Be able to handle a little inaccuracy

Pure odometry isn't the only choice!

The ideal world

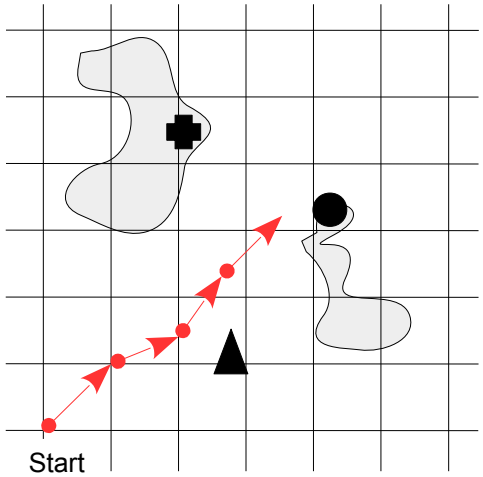
- A GPS like system that gave the position of the robot.
- GPSdoes not work at the usual scale of robots
- GPS can give you heading when you move but not the position you are facing.

Localization techniques

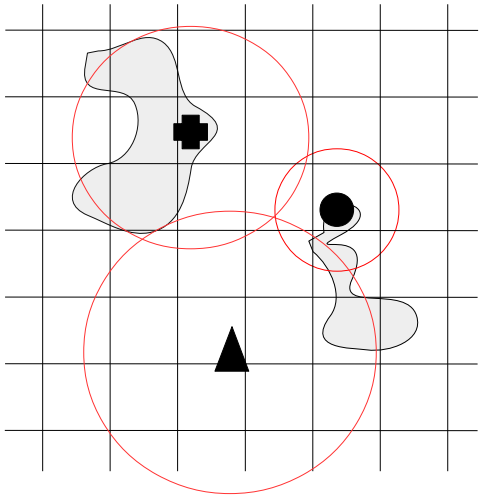
- Beacons, RFIDs, floor markings, bar codes etc
- Line following, wall following
- More/better pose providers:
  - Compass sensor, Gyro sensor, Accelerometer
  - Map based + Range sensor techniques (radar)
  - Map + Direction sensor techniques (landmarks)
- Many of these techniques works best with very predictable environments.
- No unexpected obstacles

Outside support from satellites, cameras, spotters!

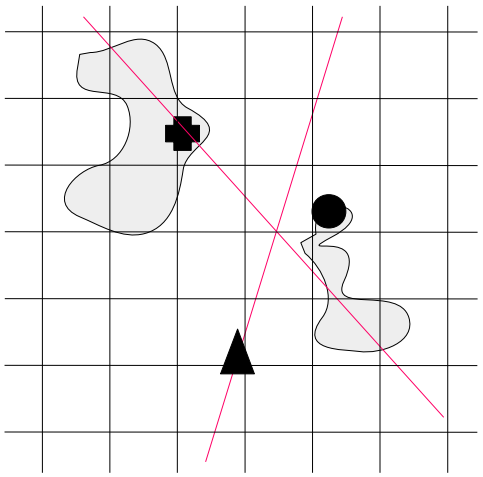
Localization techniques



<u>Dead reckoning:</u>
- Start position
- Time
- Distance
- Bearing



<u>Map + Radar/Sonar:</u>
- Map/
- Landmark locations
- Range finder



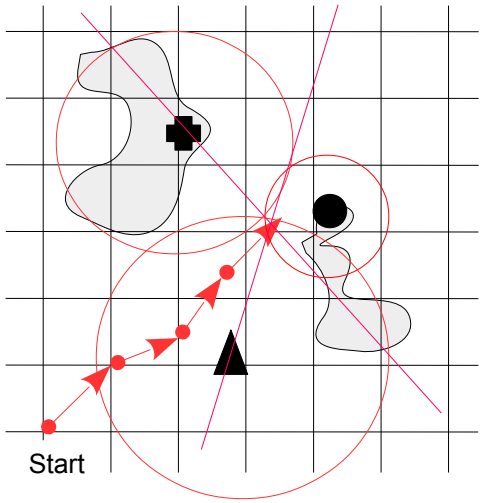
<u>Map + Compass/Gyro:</u>
- Map/
- Landmark locations
- Direction finder

Challenge

- How do you handle conflicting information?
  - Dead-reckoning says you're facing north-east.
  - The compass say you're facing north-west
  - The ultrasonic sensor detects a wall to the north consistent with a wall on the map ...
- "Sensor fusion" is a major challenge in robotics (fusing conflicting input from different sensors).

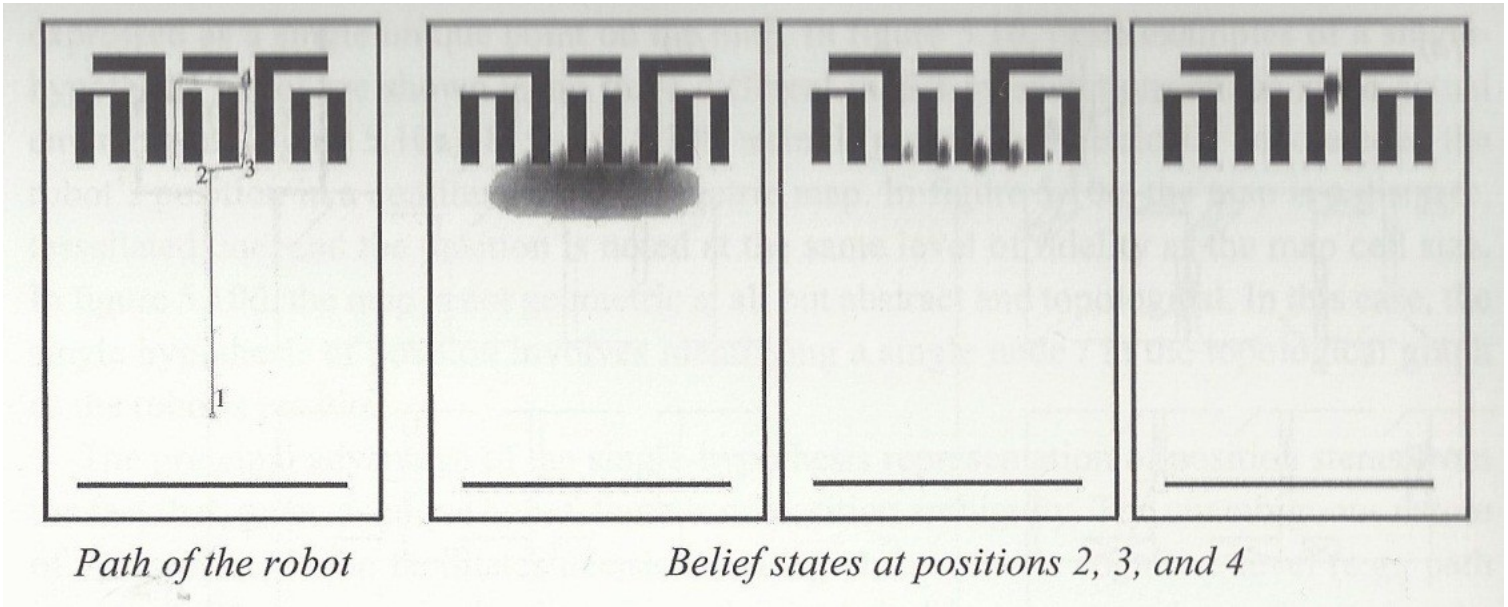
LejOS supports fusion of multiple sensors:

(Check out FusorDetector class in the API)



Single belief/multiple beliefs

- Keep several options for where you are. Refine and check when you receive more information.
- Challenge: how to decide?



Line following

- The basic principle:
- Use one light sensor and follow the left (or right) edge.
- If it gets lighter turn right
- If it get darker turn left
- If it is just right then drive straight

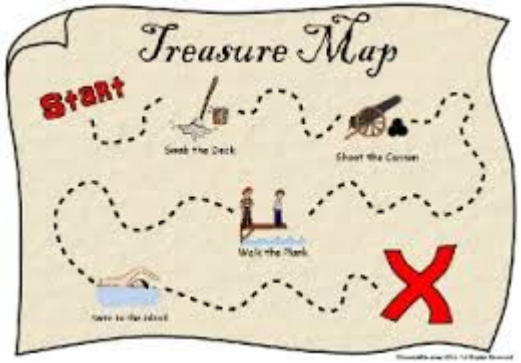
Wall following very similar: maintain distance to left or right wall. Add random moves to avoid walking in circles.

Challenges

- If you pass the line you'll keep turning right
- How do you get back on track?
- Can you automatically adjust the calibration along the way?
- Can it be expressed as behaviors?
- Is it better to express it as behaviors?

To Localize or not to localize

- Pro Localize: We may need to be able to return later. The information may be communicated to other parties.
- Against localization: The map may be outdated in a dynamic environment.



- Example: floor cleaning – typically no localization
- Get good coverage by mix of wall-following and random walks (wall- following to find narrow gaps)

Localization problems

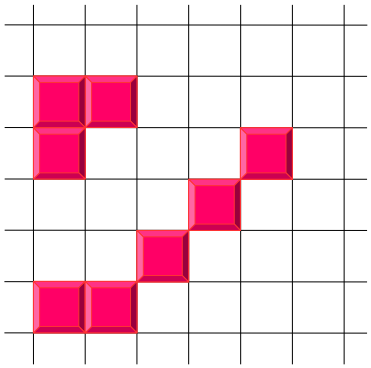
- Position tracking. Track movements, odometry
- Global positioning. Assume the robot is placed somewhere and deduce its position.
- Kidnapped robot problem.
  - If a robot is moved – how to reach the conclusion, that it has been moved and where it is now located.
- Typical problem in roboCup competitions:
  - the robot hit something and is suddenly facing the wrong direction.

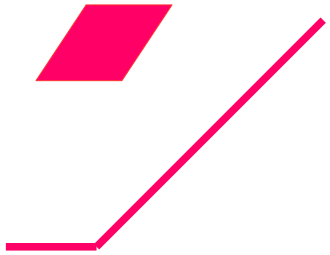
Mapping

Representing a map of an area where the robot may move.

Techniques:

- Discrete map: Make a grid and mark cells with obstacles
- Continuous maps. Obstacles are represented as lines and shapes.





Challenges: handle dynamic changes to the environment.

Techniques: use least square to fit measurements to lines and shapes.

Mapping

Representing a map in Lejos is curently only supported by a lineMap class.

LineMaps can be uploaded to system from SVG-files edited using freely available. Svg-edit is one such.

```
<svg width="132.0" height="340.0" xmlns="http://www.w3.org/2000/svg">
<g>
<line stroke="#000000" x1="32.0" x2="32.0" y1="0.0" y2="88.0"/>
<line stroke="#000000" x1="32.0" x2="0.0" y1="88.0" y2="88.0"/>
<line stroke="#000000" x1="0.0" x2="0.0" y1="88.0" y2="340.0"/>
<line stroke="#000000" x1="0.0" x2="95.0" y1="340.0" y2="340.0"/>
<line stroke="#000000" x1="95.0" x2="95.0" y1="340.0" y2="294.0"/>
<line stroke="#000000" x1="95.0" x2="132.0" y1="294.0" y2="294.0"/>
<line stroke="#000000" x1="132.0" x2="132.0" y1="294.0" y2="0.0"/>
<line stroke="#000000" x1="132.0" x2="32.0" y1="0.0" y2="0.0"/>
</g>
</svg>
```

Practice

(Exercises, Bagnall NXT chapter 13.)

- Calibrate your robot.
- Adapt NXTSlave.java (pg. 175) to the new updated navigation framework -> EV3Slave.java
- Design a small test area on the floor, measure it accurately and sketch it.
- Draw a map using SVG edit, (https://github.com/SVG-Edit/svgedit)
- Run EV3Slave with mapCommand.bat (LejosEV3/bin/)
- Experiment …, does it behave?

Advanced map uses

- Features and Feature detectors
  - Something un-mapped detected?
  - obstacle or objective?
- LejOS includes two feature detector classes, RangeDetector, and TouchDetector.
- They are sub-classes of FeatureDetectorAdapter
- More feature detectors can be fused using FusorDetcor class

```
public class FeatureAvoider {

    static final float MAX_DISTANCE = 50f;
    static final int DETECTOR_DELAY = 1000;

    public static void main(String[] args) {
        final DifferentialPilot robot = new DifferentialPilot(4.0,18.0,Motor.A, Motor.C);
        EV3IRSensor ir = new EV3IRSensor(SensorPort.S4);
        RangeFeatureDetector detector = new RangeFeatureDetector(new
RangeFinderAdapter(ir.getDistanceMode()), MAX_DISTANCE, DETECTOR_DELAY);

        detector.enableDetection(true);
        robot.forward();

        detector.addListener(new FeatureListener() {
            public void featureDetected(Feature feature, FeatureDetector detector) {
                detector.enableDetection(false);
                robot.travel(-30);
                robot.rotate(30);
                detector.enableDetection(true);
                robot.forward();
            }
        });

        while(Button.ESCAPE.isUp()) Thread.yield();
    }
}
```

Advanced map uses

Path-finding

- Given accurate map, find fastest route from A to B?

Lejos has an interface for path-finders and a number of implementing classes, e.g.:

ShortestPathFinder takes a line map as a parameter.

With a path finder, you find the path by calling the findRoute with the initial pose of the robot and the waypoint your want to reach.

A Path object is returned which can be used by the followPath method of Navigator.

Other poseProviders?

Navigator constructors in the API:

```
Navigator(MoveController pilot)
```

Allocates a Navigator object, using pilot that implements the ArcMoveController interface.

```
Navigator(MoveController pilot,
PoseProvider poseProvider)
```

Allocates a Navigator object using a pilot and a custom poseProvider, rather than the default OdometryPoseProvider

Simultaneous Localization and mapping

SLAM.

Keep information about observations and redraw map when/if the conclusion from Localization changes

Localization needs a map

Mapping requires a location

(See much more on moodle!)

Core concepts

- Sensor fusion
- Dead-reckoning
- Known world-unknown location (localization)
- Unknown world (mapping)
- Dynamic world (localization and mapping, Feature detection)
- Kidnapped robot problem

Lots of relevant and clear information on

https://lejosnews.wordpress.com/tag/navigation/

- Next week: more localization, and noise handling Monte Carlo Localization

Practice

(Exercises, Bagnall NXT chapter 14, 15.)

- Cooperate on designing larger test area, using the maze components.
- Measure it accurately, and sketch it.
- Draw a map using SVG edit,
- Adapt mapTest.java for your robot and run with mapCommand.
    - Feature detection AND/OR
    - Path finding AND/OR
- Improve poses using a EV3GyroSensor