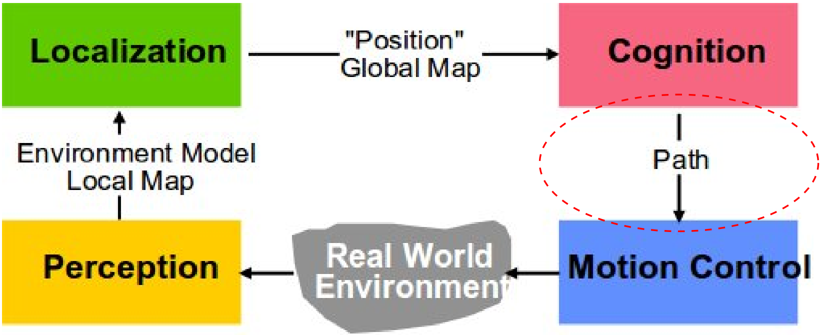
Robotics and Autonomous Systems

Lecture 13: Navigation in LeJOS

Terry Payne

Department of Computer Science
University of Liverpool

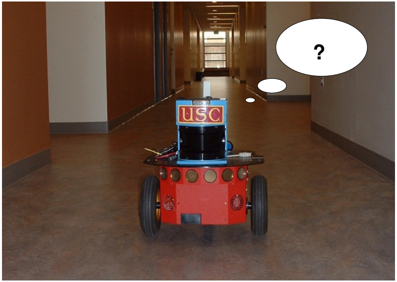
Today



- From navigation to motion control.

Navigation and path execution

- We started this course with three questions:



    - Where am I ?
    - Where am I going ?
    - How do I get there ?

- Digging into the detail of how you do the last two.

Navigation and path execution

- Last time convered how to go from:
  - Map
  - Start point
  - End point

  to a sequence of waypoints that the robot has to traverse to get from the start to the goal.
- Turns out LeJOS has quite a lot of support for this.
- LeJOS can also help with the business of following the waypoints.

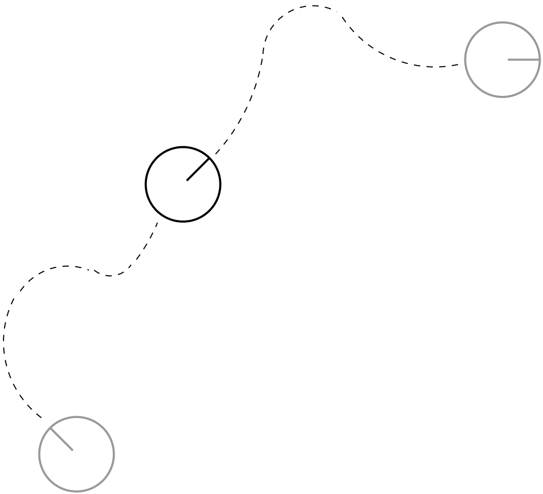
Path following

- The planning methods we covered last time returned a sequence of waypoints.
- Sequence of robot poses:

$$
\begin{aligned}
&(x_s, y_s, \theta_s)\\
&(x_1, y_1, \theta_1)\\
&\vdots\\
&(x_g, y_g, \theta_g)
\end{aligned}
$$

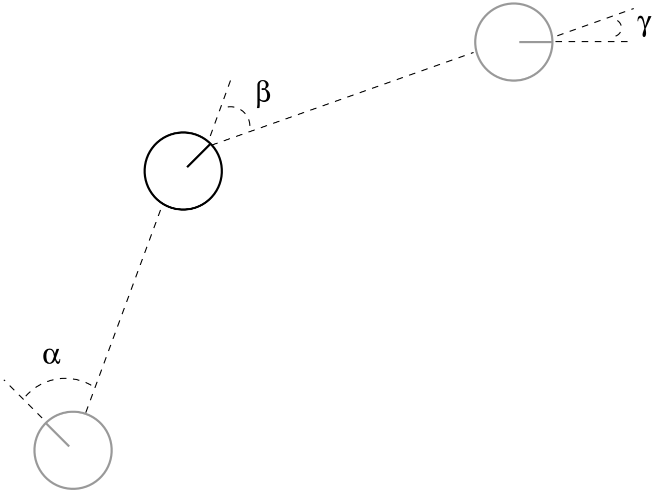
- Robot then needs to know how to move between the points.

Path following



- Plan doesn't restrict movement between waypoints.

Path following



- Simplest route is to turn to face the next point, then drive straight.

Path following

- In this case we need to:
  - Rotate α
  - Drive
  - Rotate β
  - Drive
  - Rotate γ
- Not trivial to figure out the rotations required.
- Distances are pretty easy using the Euclidian distance formula from the previous lecture.

Navigator

Amateur Navigation

- LeJOS supports this through the `Navigator` class.

LeJOS `Navigator`

- Provides all the functionality you need to have the robot follow a sequence of waypoints.
- Controls the robot using our old friend:
    - `DifferentialPilot`

  with all the good and bad things that entails.
- Tracks where the robot is currently using:
    - `PoseProvider`
- Default is `OdometryPoseProvider`
- But you can use `MCLPoseProvider` also

- The pilot allows it to control the robot

  …by sending it rotations and translations.
- The pose provider allows it to keep track of where it is.

  …so it can compute what movements are necessary.

LeJOS `Navigator`

- `Navigator` can be fed a sequence of waypoints.
- Or it can be fed a path.

Waypoint

- LeJOS has an object to represent waypoints.
- A waypoint can be a point:
  - `Point` object
  - *x* and *y* coordinate.

  or it can be a pose:
  - `Pose` object
  - *x* and *y* coordinate, plus a heading.
- A waypoint that is a point has heading 0.

Waypoint

- API for waypoint includes the method:

  ```
  boolean checkValidity(Pose p)
  ```

  which checks if the pose is close enough to the waypoint to count as having reached it.
- So, you can `extend` the class and override the function to create your own waypoint that is reached approximately.

Path

- `Path` is an ordered sequence of `Waypoints`.
- Implemented as a Java `ArrayList` of `Waypoints`
- But you don't need to manipulate it.

Navigator example

- Key functions are the following:
- `Navigator(MoveController p)`
  Constructor, takes a pilot object as an argument.
  The navigator then uses this to drive the robot.
- ```
  void addWayPoint(float x, float y)
  void addWayPoint(float x, float y,
                   float heading)
  ```
  Adds a waypoint to the path.
- `void followPath()`
  Drive to each waypoint in the path, in turn.
  ```

Navigator example

- That's all you need to use the `Navigator`

Navigator example

- My code (below) uses a couple of additional functions, mainly for show.
- `boolean pathCompleted()`
  Reports if the robot has completed the current path.
- `Waypoint getWaypoint()`
  Returns the current waypoint, the one the robot is heading to.

Navigator example

```
public class PathFollower{
  private DifferentialPilot pilot
    = new DifferentialPilot(3.25, 19.8, Motor.C, Motor.B);
  private Navigator navigator = new Navigator(pilot);
  private Waypoint next;

  public void newWaypoint(int x, int y){
    navigator.addWaypoint(x, y);
  }

  public void navigate(){
    while(!navigator.pathCompleted()){
    navigator.followPath();
    next = navigator.getWaypoint();
    LCD.drawString("Moving to...", 0, 0);
    LCD.drawString("(" + (int)next.getX() +
      "," + (int)next.getY() + ")", 0, 1);
    }
  }
}
```

Navigator example

- Note the use of private members for the `PathFollower` class.
- These, naturally, need public access functions.
- IMHO, this is good Java (and OO) style.

Navigator example

- The navigator runs its own thread, so the `while` loop is really just for driving the LCD display.
- Should work perfectly well as:

```
public void navigate(){
  navigator.followPath();
  while(!navigator.pathCompleted()){
    next = navigator.getWaypoint();
    LCD.drawString("Moving to...", 0, 0);
    LCD.drawString("(" + (int)next.getX() +
      "," + (int)next.getY() + ")", 0, 1);
  }
```

(though I haven't tested this version).

Navigator example

- The `main()` is in another class, which creates a `Pathfinder` object, and sets up the waypoints.

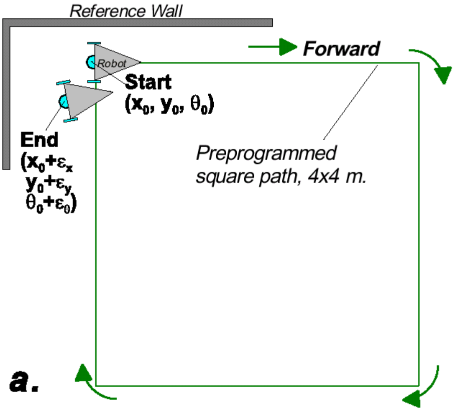
```
public class RunNavigator{
  public static void main(String[] args)
      throws Exception{
    PathFollower pFollow = new PathFollower();

    pFollow.newWaypoint(40, 0);
    pFollow.newWaypoint(40, 40);
    pFollow.newWaypoint(0, 40);
    pFollow.newWaypoint(0, 0);

    pFollow.navigate();
  }
}
```

Navigator example

- This is another version of the “drive in a square” program.



Navigator example

- Both these classes are available from the module website.

Path finding

- What we have so far is a program that will work when it already knows what the waypoints are.
- Not much use if you don't don't them.
- But LeJOS can help with this too.

LineMap

- To do any pathfinding, we need a map, and LeJOS gives us the `LineMap` class.
- We set up a `LineMap` like this:

```
Line[] lines = new Line[3];
lines[0] = new Line(50, 100, 100, 100);
lines[1] = new Line(100, 100, 100, 50);
lines[2] = new Line(50, 100, 100, 50);

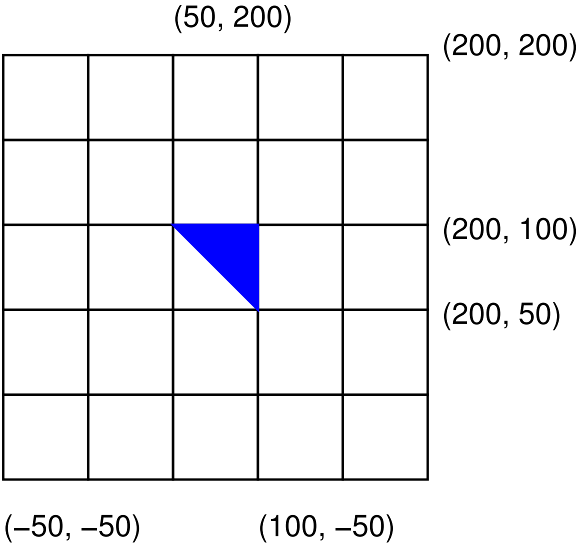

Rectangle bounds = new Rectangle(-50, -50, 200, 200);

LineMap map = new LineMap(lines, bounds);
```

LineMap

- A LineMap is an array of lines and a bounding box.
- A bounding box is just a rectangle, specified by the:
    - bottom left
      (smallest x and y coordinates)
    - top right
      (largest x and y coordinates)
- Each line is defined by two pairs of coordinates that mark the end points of the line.

LineMap



PathFinder

- Then all we have to do is:

```
Pose start = new Pose(0, 0, 270);
Waypoint goal = new Waypoint(125, 150);

ShortestPathFinder finder
  = new ShortestPathFinder(map);
finder.lengthenLines(5);

PathFollower pFollow = new PathFollower();
pFollow.newPose(start);

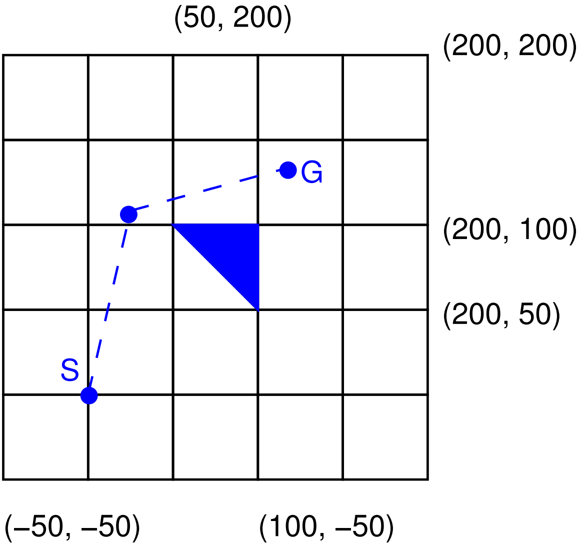
Path path = finder.findRoute(start, goal);
pFollow.newPath(path);
pFollow.navigate();
```

PathFinder

- Set up start and end points.
  For some reason they are different types of object.
- Create a path finder object and give it the map.
  - `ShortestPathFinder`
  - `DijkstraPathFinder`

  Not clear what the difference is between these.
- Create clearance for the robot
  Basic path finder only works for infinitely small robots/points.
- Create an instance of our `PathFollower` class.
- Call the PathFinder on the start and goal points, and get a path back.
- Pass the path to the `PathFollower` and let it use its navigator to follow it.

PathFinder

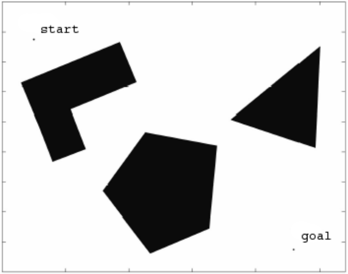


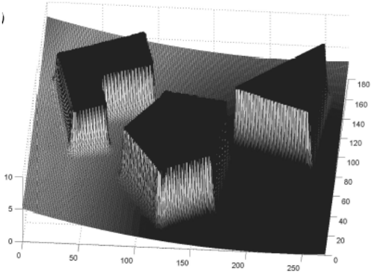
Other forms of navigation

- Having seen how LeJOS can implement path finding in a map, let's look at a couple of other forms of navigation that we might implement on the NXT.
- Simpler in some ways.
- Not necessarily as good at getting to the goal.

Potential field

- Robot is treated as a point under the influence of an artificial potential field.



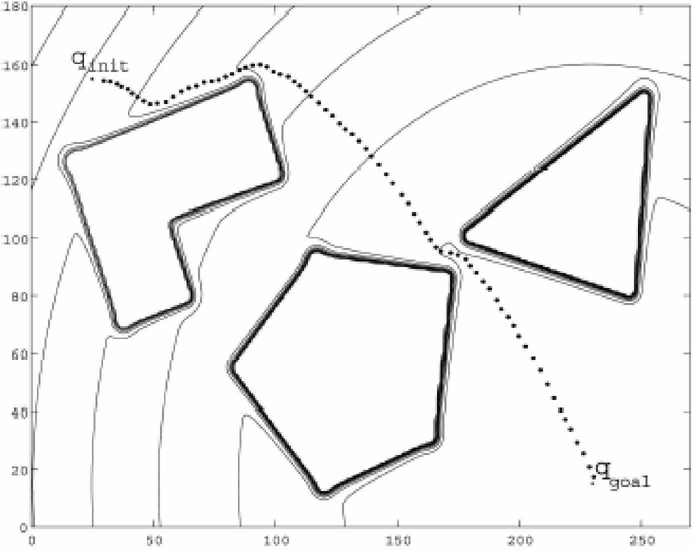


- The `goal` attracts it and obstacles repel it.

Potential field

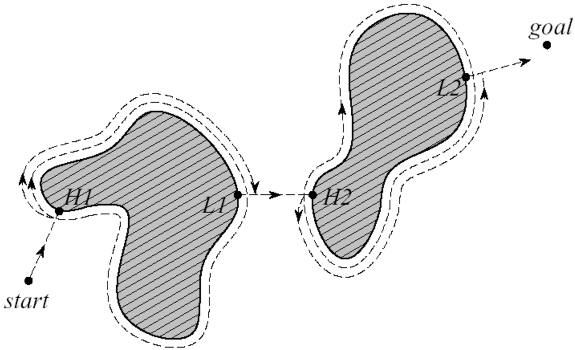
- If you know your location and the goal, can compute the “force” on the robot.
- Navigator will make direct the robot towards the goal point.
  - Just make it the next waypoint.
- Obstacle avoidance provides the “repulsion” from obstacles.
  - Steer a path that is a combination of repulsion from obstacle and heading to goal.
  - This will require taking control back from the Navigator.
- ‘Don’t need a map.

Potential field



Bug algorithms

- The bug algorithms assume localization but no map.



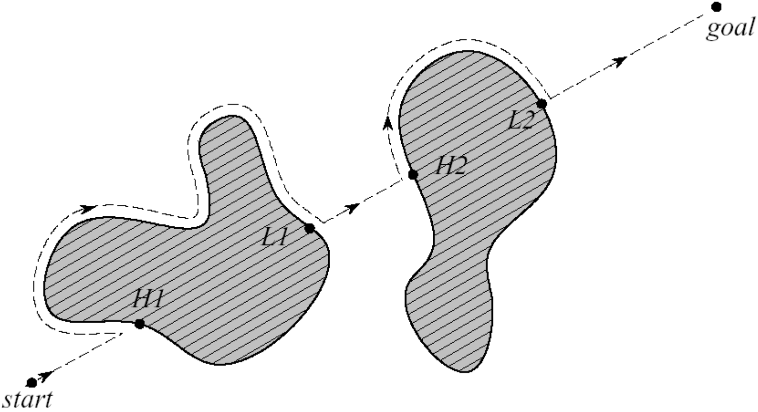
- Here we see the first such algorithm, bug 1, working.

Bug algorithms

- When you meet an obstacle you follow around the edge.
- Leave the obstacle at the point closest to the goal.
- Circle the obstacle to be sure that you know where this point is.

Bug algorithms

- Here's the second bug algorithm in action.



- Improves on the performance of bug 1

Bug algorithms

- Follow the obstacle always on the left or right side.
- Leave the obstacle if you cross the direct (line of sight) connection between start and goal.

Summary

- Today we looked more at navigation.
- Primarily we looked at the support LeJOS provides for navigation:
  - Path following
  - Path finding

  which make it possible to carry out the kind of navigation discussed in Lecture 12.
- We also looked at a couple of additional navigation techniques.