Introduction to Robotics

Bug Algorithms

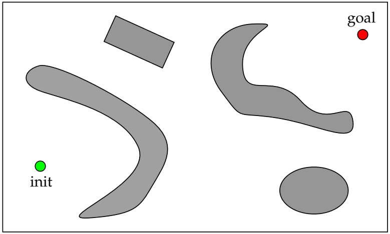
Erion Plaku

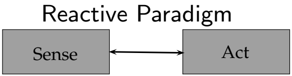
Department of Electrical Engineering and Computer Science
Catholic University of America

Outline

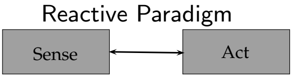
1. General Properties of Bug Path-Planning Algorithms
2. Bug Algorithms with Tactile (Contact) Sensors
   - Bug0
   - Bug1
   - Bug2
3. Bug Algorithms with Range Sensors
   - TangentBug
4. Summary

Problem: Compute a collision-free path from an initial to a goal position





- No global model of the world, i.e., obstacles are unknown
- Only local information acquired through sensing
- Inspired by insects

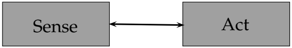


- No global model of the world, i.e., obstacles are unknown
- Only local information acquired through sensing
- Inspired by insects

Properties

- Complete algorithms, i.e., find solution if it exists, report no when there is no solution
- Theoretical lower and upper bounds on path length; optimal paths in certain cases

Reactive Paradigm



- No global model of the world, i.e., obstacles are unknown
- Only local information acquired through sensing
- Inspired by insects

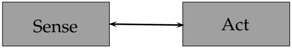
Properties

- Complete algorithms, i.e., find solution if it exists, report no when there is no solution
- Theoretical lower and upper bounds on path length; optimal paths in certain cases

Environment

- Two-dimensional scene filled with unknown obstacles
- Each obstacle is a simple closed curve of finite length and non-zero thickness
- A straight line crosses an obstacle finitely many times
- Obstacles do not touch each other
- Locally finite number of obstacles, i.e., any disc of finite radius intersects a finite set of obstacles
- Initial and goal positions are known

Reactive Paradigm



- No global model of the world, i.e., obstacles are unknown
- Only local information acquired through sensing
- Inspired by insects

Properties

- Complete algorithms, i.e., find solution if it exists, report no when there is no solution
- Theoretical lower and upper bounds on path length; optimal paths in certain cases

Environment

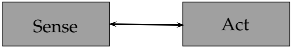
- Two-dimensional scene filled with unknown obstacles
- Each obstacle is a simple closed curve of finite length and non-zero thickness
- A straight line crosses an obstacle finitely many times
- Obstacles do not touch each other
- Locally finite number of obstacles, i.e., any disc of finite radius intersects a finite set of obstacles
- Initial and goal positions are known

Point Robot, Simple Motions

- Move straight toward `goal`
- Move along obstacle boundary
- Stop

Bug Path-Planning Algorithms

Reactive Paradigm



- No global model of the world, i.e., obstacles are unknown
- Only local information acquired through sensing
- Inspired by insects

Properties

- Complete algorithms, i.e., find solution if it exists, report no when there is no solution
- Theoretical lower and upper bounds on path length; optimal paths in certain cases

Environment

- Two-dimensional scene filled with unknown obstacles
- Each obstacle is a simple closed curve of finite length and non-zero thickness
- A straight line crosses an obstacle finitely many times
- Obstacles do not touch each other
- Locally finite number of obstacles, i.e., any disc of finite radius intersects a finite set of obstacles
- Initial and goal positions are known

Point Robot, Simple Motions

- Move straight toward goal
- Move along obstacle boundary
- Stop

Simple Sensing

- `Bug1`, `Bug2` assume essentially tactile (contact) sensing
- `TangentBug`, `VisBug`, `DistBug` deal with finite distance sensing
- `I-Bug` uses only signal strength emanating from goal

Bug with Tactile (Contact) Sensor

Tactile Sensor

- Provides current position
- Detects when a contact with an obstacle occurs

Bug with Tactile (Contact) Sensor

Tactile Sensor

- Provides current position
- Detects when a contact with an obstacle occurs

`Bug0`, `Bug1`, `Bug2` Algorithms – General Idea

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow obstacle boundary
  - at some point, leave the obstacle and head again toward `goal`

Bug with Tactile (Contact) Sensor

Tactile Sensor

- Provides current position
- Detects when a contact with an obstacle occurs

`Bug0`, `Bug1`, `Bug2` Algorithms – General Idea

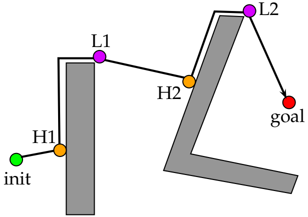
repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow obstacle boundary
  - at some point, leave the obstacle and head again toward `goal`

Path consists of a sequence of hit (H_i) and leave (L_i) points
Algorithms differ on how leave points are computed

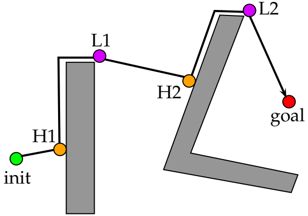
repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow obstacle boundary
    until can head toward `goal` again



repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow obstacle boundary
    until can head toward `goal` again

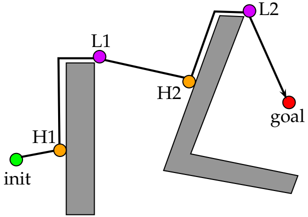


Is Bug0 a complete algorithm? Can you think of a scenario where Bug0 fails to find a solution even though a solution exists?

Bug0 Algorithm

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow obstacle boundary
    until can head toward `goal` again

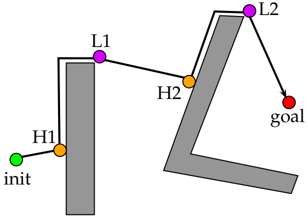


Is `Bug0` a complete algorithm? Can you think of a scenario where `Bug0` fails to find a solution even though a solution exists?

Problem is that `Bug0` has no memory

repeat until goal is reached

- head toward goal
- if sensor reports contact with an obstacle then
  - follow obstacle boundary
    until can head toward goal again



Is Bug0 a complete algorithm? Can you think of a scenario where Bug0 fails to find a solution even though a solution exists?

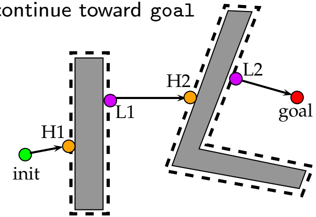
Problem is that Bug0 has no memory

Can we obtain a complete algorithm if Bug has some memory?

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

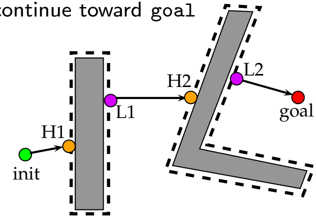
- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



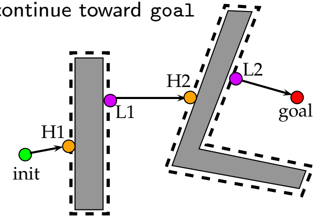
Bug1 Pseudocode

1: $L_0 \leftarrow$ `init`; $i \leftarrow 1$
2: **loop**

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



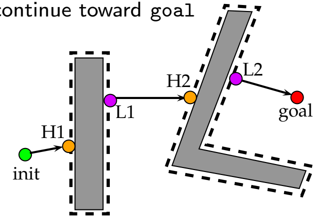
Bug1 Pseudocode

1: $L_0 \leftarrow$ `init`; $i \leftarrow 1$
2: **loop**
3: **repeat** move on a straight line from L_{i-1} to `goal`
4: **until** `goal` is reached or obstacle is encountered at H_i

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



Bug1 Pseudocode

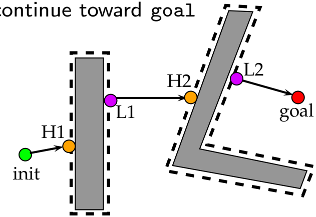
1: $L_0 \leftarrow$ `init`; $i \leftarrow 1$
2: **loop**
3: **repeat** move on a straight line from L_{i-1} to `goal`
4: **until** `goal` is reached or obstacle is encountered at H_i
5: **if** `goal` is reached **then** exit with success

Bug1 Algorithm

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



Bug1 Pseudocode

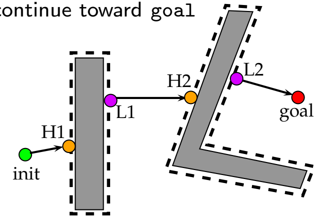
$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
  repeat move on a straight line from L_{i-1} to `goal`
  until `goal` is reached or obstacle is encountered at H_i
  if `goal` is reached **then** exit with success
  repeat follow boundary recording point L_i with shortest distance to `goal`
  until `goal` is reached or H_i is re-encountered

Bug1 Algorithm

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



Bug1 Pseudocode

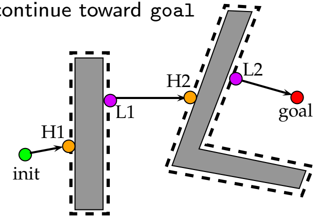
$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
  repeat move on a straight line from L_{i-1} to `goal`
  until `goal` is reached or obstacle is encountered at H_i
  if `goal` is reached **then** exit with success
  repeat follow boundary recording point L_i with shortest distance to `goal`
  until `goal` is reached or H_i is re-encountered
  if `goal` is reached **then** exit with success

Bug1 Algorithm

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



Bug1 Pseudocode

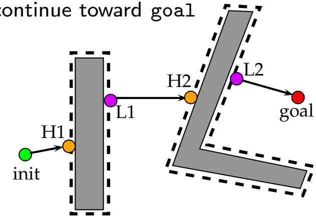
```
L_0 ← init; i ← 1
loop
    repeat move on a straight line from L_{i−1} to goal
    until goal is reached or obstacle is encountered at H_i
    if goal is reached then exit with success
    repeat follow boundary recording point L_i with shortest distance to goal
    until goal is reached or H_i is re-encountered
    if goal is reached then exit with success
    follow boundary from H_i to L_i along shortest route
```

Bug1 Algorithm

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - circumnavigate the obstacle and *remember* how close you get to the goal
  - return to that closest point (by wall following) and continue toward `goal`



Bug1 Pseudocode

$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
    repeat move on a straight line from L_{i-1} to `goal`
    until `goal` is reached or obstacle is encountered at H_i
    if `goal` is reached **then** exit with success
    repeat follow boundary recording point L_i with shortest distance to `goal`
    until `goal` is reached or H_i is re-encountered
    if `goal` is reached **then** exit with success
    follow boundary from H_i to L_i along shortest route
    if move on straight line from L_i toward `goal` moves into obstacle **then** exit with failure
    else $i \leftarrow i + 1$

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch:

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`, the bug never returns to this obstacle again

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to `goal`
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward $\mathtt{goal}$, the bug never returns to this obstacle again

Proof Sketch: Consider the sequence of points visited by bug: $\mathtt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \mathtt{goal}) \geq d(L_i, \mathtt{goal})$ since L_i closest point on obstacle boundary to $\mathtt{goal}$
- $d(H_i, \mathtt{goal}) > d(L_i, \mathtt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore,

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\mathtt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \mathtt{goal}) \geq d(L_i, \mathtt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \mathtt{goal}) > d(L_i, \mathtt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \mathtt{goal}) > d(H_{i+1}, \mathtt{goal})$ since different obstacles do not touch

Therefore, $d(\mathtt{init}, \mathtt{goal}) \geq d(H_1, \mathtt{goal}) > d(L_1, \mathtt{goal}) > d(H_2, \mathtt{goal}) > d(L_2, \mathtt{goal}) > \ldots$

Thus,

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward goal, the bug never returns to this obstacle again

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
    - if straight line is tangent to obstacle, then no circumnavigation
    - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore, $d(\texttt{init}, \texttt{goal}) \geq d(H_1, \texttt{goal}) > d(L_1, \texttt{goal}) > d(H_2, \texttt{goal}) > d(L_2, \texttt{goal}) > \ldots$

Thus, since $d(L_i, \texttt{goal})$ is the shortest distance from the i-th obstacle to goal and since each each new hit point is closer than the last leave point, then bug cannot encounter the i-th obstacle again

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore, $d(\texttt{init}, \texttt{goal}) \geq d(H_1, \texttt{goal}) > d(L_1, \texttt{goal}) > d(H_2, \texttt{goal}) > d(L_2, \texttt{goal}) > \ldots$

Thus, since $d(L_i, \texttt{goal})$ is the shortest distance from the i-th obstacle to goal and since each each new hit point is closer than the last leave point, then bug cannot encounter the i-th obstacle again

Lemma 2: Bug meets only a finite number of obstacles

Proof Sketch:

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore, $d(\texttt{init}, \texttt{goal}) \geq d(H_1, \texttt{goal}) > d(L_1, \texttt{goal}) > d(H_2, \texttt{goal}) > d(L_2, \texttt{goal}) > \ldots$

Thus, since $d(L_i, \texttt{goal})$ is the shortest distance from the i-th obstacle to goal and since each each new hit point is closer than the last leave point, then bug cannot encounter the i-th obstacle again

Lemma 2: Bug meets only a finite number of obstacles

Proof Sketch: Straight-line segments from L_i to H_{i+1} $(i = 0, 1, \ldots)$ are within the same circle of radius $d(\texttt{init}, \texttt{goal})$ centered at goal since

Bug1 Properties

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore, $d(\texttt{init}, \texttt{goal}) \geq d(H_1, \texttt{goal}) > d(L_1, \texttt{goal}) > d(H_2, \texttt{goal}) > d(L_2, \texttt{goal}) > \ldots$

Thus, since $d(L_i, \texttt{goal})$ is the shortest distance from the i-th obstacle to goal and since each each new hit point is closer than the last leave point, then bug cannot encounter the i-th obstacle again

Lemma 2: Bug meets only a finite number of obstacles

Proof Sketch: Straight-line segments from L_i to H_{i+1} $(i = 0, 1, \ldots)$ are within the same circle of radius $d(\texttt{init}, \texttt{goal})$ centered at goal since

- each hit point is closer than the last leave point

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward goal, the bug never returns to this obstacle again

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore, $d(\texttt{init}, \texttt{goal}) \geq d(H_1, \texttt{goal}) > d(L_1, \texttt{goal}) > d(H_2, \texttt{goal}) > d(L_2, \texttt{goal}) > \ldots$

Thus, since $d(L_i, \texttt{goal})$ is the shortest distance from the i-th obstacle to goal and since each each new hit point is closer than the last leave point, then bug cannot encounter the i-th obstacle again

Lemma 2: Bug meets only a finite number of obstacles

Proof Sketch: Straight-line segments from L_i to H_{i+1} $(i = 0, 1, \ldots)$ are within the same circle of radius $d(\texttt{init}, \texttt{goal})$ centered at goal since

- each hit point is closer than the last leave point
- assumption that any finite disc can intersect only a finite number of obstacles

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward `goal`*, the bug never returns to this obstacle again*

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore, $d(\texttt{init}, \texttt{goal}) \geq d(H_1, \texttt{goal}) > d(L_1, \texttt{goal}) > d(H_2, \texttt{goal}) > d(L_2, \texttt{goal}) > \ldots$
Thus, since $d(L_i, \texttt{goal})$ is the shortest distance from the i-th obstacle to goal and since each each new hit point is closer than the last leave point, then bug cannot encounter the i-th obstacle again

Lemma 2: Bug meets only a finite number of obstacles

Proof Sketch: Straight-line segments from L_i to H_{i+1} $(i = 0, 1, \ldots)$ are within the same circle of radius $d(\texttt{init}, \texttt{goal})$ centered at goal since

- each hit point is closer than the last leave point
- assumption that any finite disc can intersect only a finite number of obstacles

Corollary: `Bug1` *algorithm always terminates in finite time*

Lemma 1: When the bug leaves a leave point of an obstacle to continue its way toward goal, the bug never returns to this obstacle again

Proof Sketch: Consider the sequence of points visited by bug: $\texttt{init}, H_1, L_1, H_2, L_2, \ldots$

- $d(H_i, \texttt{goal}) \geq d(L_i, \texttt{goal})$ since L_i closest point on obstacle boundary to goal
- $d(H_i, \texttt{goal}) > d(L_i, \texttt{goal})$ since $H_i \neq L_i$. Why?
  - if straight line is tangent to obstacle, then no circumnavigation
  - otherwise, straight line crosses obstacle at two distinct points (since obstacle has finite thickness)
- $d(L_i, \texttt{goal}) > d(H_{i+1}, \texttt{goal})$ since different obstacles do not touch

Therefore, $d(\texttt{init}, \texttt{goal}) \geq d(H_1, \texttt{goal}) > d(L_1, \texttt{goal}) > d(H_2, \texttt{goal}) > d(L_2, \texttt{goal}) > \ldots$

Thus, since $d(L_i, \texttt{goal})$ is the shortest distance from the i-th obstacle to goal and since each each new hit point is closer than the last leave point, then bug cannot encounter the i-th obstacle again

Lemma 2: Bug meets only a finite number of obstacles

Proof Sketch: Straight-line segments from L_i to H_{i+1} ($i = 0, 1, \ldots$) are within the same circle of radius $d(\texttt{init}, \texttt{goal})$ centered at goal since

- each hit point is closer than the last leave point
- assumption that any finite disc can intersect only a finite number of obstacles

Corollary: Bug1 algorithm always terminates in finite time

Proof Sketch: Follows immediately from Lemma 1 and Lemma 2

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch:

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1`

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But. . .

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But...

1. Lemma 1 and 2 imply that `Bug1` always terminates in finite time

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But. . .

1. Lemma 1 and 2 imply that `Bug1` always terminates in finite time
2. `Bug1` never goes through an obstacle, so it only computes valid paths. So, if `Bug1` reports finding a path to `goal`, then there is a path to `goal`

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But...

1. Lemma 1 and 2 imply that `Bug1` always terminates in finite time
2. `Bug1` never goes through an obstacle, so it only computes valid paths. So, if `Bug1` reports finding a path to `goal`, then there is a path to `goal`
3. Then, move from last leave point toward `goal` crosses into obstacle

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But. . .

1. Lemma 1 and 2 imply that `Bug1` always terminates in finite time
2. `Bug1` never goes through an obstacle, so it only computes valid paths. So, if `Bug1` reports finding a path to `goal`, then there is a path to `goal`
3. Then, move from last leave point toward `goal` crosses into obstacle
   - But, line must cross obstacle even number of times (Jordan curve theorem)

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But...

1. Lemma 1 and 2 imply that `Bug1` always terminates in finite time
2. `Bug1` never goes through an obstacle, so it only computes valid paths. So, if `Bug1` reports finding a path to `goal`, then there is a path to `goal`
3. Then, move from last leave point toward `goal` crosses into obstacle
   - But, line must cross obstacle even number of times (Jordan curve theorem)
   - Then, there is another intersection point on boundary closer to `goal`

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But...

1. Lemma 1 and 2 imply that `Bug1` always terminates in finite time
2. `Bug1` never goes through an obstacle, so it only computes valid paths. So, if `Bug1` reports finding a path to `goal`, then there is a path to `goal`
3. Then, move from last leave point toward `goal` crosses into obstacle
   - But, line must cross obstacle even number of times (Jordan curve theorem)
   - Then, there is another intersection point on boundary closer to `goal`
   - Since, we assumed there is a path to `goal`, then `goal` cannot be encircled by obstacle

Bug1 Completeness Analysis

Theorem: `Bug1` is a complete path-planning algorithm, i.e., in finite time, `Bug1`

- *finds a path to `goal` when a path exists or*
- *terminates with failure when there is no path to `goal`*

Proof Sketch: Assume to the contrary that `Bug1` is incomplete. Then

1. `Bug1` does not terminate in finite time, or
2. There is no path to `goal`, but `Bug1` incorrectly reports finding a path, or
3. There is at least a path to `goal`, but `Bug1` incorrectly reports finding no path

But...

1. Lemma 1 and 2 imply that `Bug1` always terminates in finite time
2. `Bug1` never goes through an obstacle, so it only computes valid paths. So, if `Bug1` reports finding a path to `goal`, then there is a path to `goal`
3. Then, move from last leave point toward `goal` crosses into obstacle
   - But, line must cross obstacle even number of times (Jordan curve theorem)
   - Then, there is another intersection point on boundary closer to `goal`
   - Since, we assumed there is a path to `goal`, then `goal` cannot be encircled by obstacle
   - Thus, bug must have encountered this other intersection point (which is supposedly closer to the goal) when circumnavigating obstacle boundary, which contradicts definition of leave point

Lower Bound: What is the shortest distance that Bug1 might travel?

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq$

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq d(\texttt{init}, \texttt{goal})$. Why?

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq d(\texttt{init}, \texttt{goal})$. Why? (leave point is closest to obstacle)

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq d(\texttt{init}, \texttt{goal})$. Why? (leave point is closest to obstacle)
- when going from H_i to L_i, Bug1 first circumnavigates the i-th obstacle and then, after coming back to H_i, selects the shorter route to go to L_i. Thus, $1.5 p_i$, where p_i is the perimeter of the i-th obstacle

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq d(\texttt{init}, \texttt{goal})$. Why? (leave point is closest to obstacle)
- when going from H_i to L_i, Bug1 first circumnavigates the i-th obstacle and then, after coming back to H_i, selects the shorter route to go to L_i. Thus, $1.5p_i$, where p_i is the perimeter of the i-th obstacle

Therefore, upper bound

$$d(\texttt{init}, \texttt{goal}) + 1.5 \sum_{i=1}^{n} p_i$$

What is n?

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq d(\texttt{init}, \texttt{goal})$. Why? (leave point is closest to obstacle)
- when going from H_i to L_i, Bug1 first circumnavigates the i-th obstacle and then, after coming back to H_i, selects the shorter route to go to L_i. Thus, $1.5p_i$, where p_i is the perimeter of the i-th obstacle

Therefore, upper bound

$$d(\texttt{init}, \texttt{goal}) + 1.5 \sum_{i=1}^{n} p_i$$

What is n?

- number of obstacles intersecting the disc of radius $d(\texttt{init}, \texttt{goal})$ centered at goal

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq d(\texttt{init}, \texttt{goal})$. Why? (leave point is closest to obstacle)
- when going from H_i to L_i, Bug1 first circumnavigates the i-th obstacle and then, after coming back to H_i, selects the shorter route to go to L_i. Thus, $1.5p_i$, where p_i is the perimeter of the i-th obstacle

Therefore, upper bound

$$d(\texttt{init}, \texttt{goal}) + 1.5 \sum_{i=1}^{n} p_i$$

What is n?

- number of obstacles intersecting the disc of radius $d(\texttt{init}, \texttt{goal})$ centered at goal

Remind me again why it is not necessary to consider obstacles outside this disk?

Bug1 Lower and Upper Bounds on Path Length

Lower Bound: What is the shortest distance that Bug1 might travel?

- $d(\texttt{init}, \texttt{goal})$ (straight-line to goal, no obstacles encountered)

Upper Bound: What is an upper bound on the longest distance that Bug1 might travel?

- any path can be looked as consisting of straight-line segments (from L_{i_1} to H_i) and walking around the obstacles
- sum of straight-line segments $\leq d(\texttt{init}, \texttt{goal})$. Why? (leave point is closest to obstacle)
- when going from H_i to L_i, Bug1 first circumnavigates the i-th obstacle and then, after coming back to H_i, selects the shorter route to go to L_i. Thus, $1.5 p_i$, where p_i is the perimeter of the i-th obstacle

Therefore, upper bound

$$d(\texttt{init}, \texttt{goal}) + 1.5 \sum_{i=1}^{n} p_i$$

What is n?

- number of obstacles intersecting the disc of radius $d(\texttt{init}, \texttt{goal})$ centered at goal

Remind me again why it is not necessary to consider obstacles outside this disk?

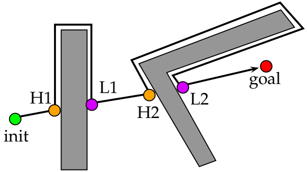
- see proof of Lemma 2, distances from $H_1, L_1, H_2, L_2, \ldots$ to goal become smaller and smaller and are never more than $d(\texttt{init}, \texttt{goal})$. So, bug never encounters obstacles outside this disk

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

call the line from `init` to `goal` the *m*-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the *m*-line again
  - leave the obstacle and continue straight toward `goal`



Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

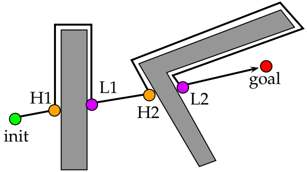
call the line from `init` to `goal` the *m*-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the *m*-line again
  - leave the obstacle and continue straight toward `goal`

Bug2 Pseudocode

$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
  repeat move on a straight line from L_{i-1} to `goal`
  until `goal` is reached or obstacle is encountered at H_i
  if `goal` is reached **then** exit with success



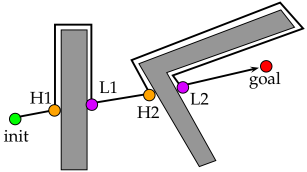
Bug2 Algorithm

Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

call the line from `init` to `goal` the *m*-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the *m*-line again
  - leave the obstacle and continue straight toward `goal`



Bug2 Pseudocode

$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
  repeat move on a straight line from L_{i-1} to `goal`
  until `goal` is reached or obstacle is encountered at H_i
  if `goal` is reached **then** exit with success
  repeat follow boundary
  until
  (a) `goal` is reached or

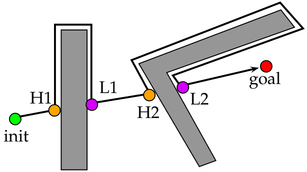
Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

call the line from `init` to `goal` the *m*-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the *m*-line again
  - leave the obstacle and continue straight toward `goal`

Bug2 Pseudocode



$L_0 \leftarrow$ init; $i \leftarrow 1$
loop
    repeat move on a straight line from L_{i-1} to goal
    until goal is reached or obstacle is encountered at H_i
    if goal is reached **then** exit with success
    repeat follow boundary
    until
    (a) goal is reached or
    (b) *m*-line is re-encountered at Q such that $Q \neq H_i$, $d(Q, \texttt{goal}) < d(H_i, \texttt{goal})$, and line $(Q, \texttt{goal})$ does not cross the current obstacle at Q or

Bug2 Algorithm

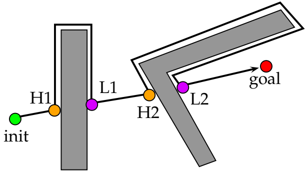
Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

call the line from `init` to `goal` the *m*-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the *m*-line again
  - leave the obstacle and continue straight toward `goal`

Bug2 Pseudocode



$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
  repeat move on a straight line from L_{i-1} to `goal`
  until `goal` is reached or obstacle is encountered at H_i
  if `goal` is reached **then** exit with success
  repeat follow boundary
  until
    (a) `goal` is reached or
    (b) *m*-line is re-encountered at Q such that $Q \neq H_i$, $d(Q, \texttt{goal}) < d(H_i, \texttt{goal})$, and line $(Q, \texttt{goal})$ does not cross the current obstacle at Q or
    (c) H_i is re-encountered

Bug2 Algorithm

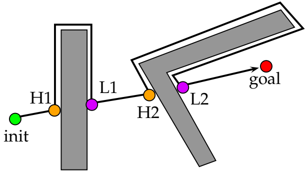
Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

call the line from `init` to `goal` the m-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the m-line again
  - leave the obstacle and continue straight toward `goal`

Bug2 Pseudocode



$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
  repeat move on a straight line from L_{i-1} to `goal`
  until `goal` is reached or obstacle is encountered at H_i
  if `goal` is reached **then** exit with success
  repeat follow boundary
  until
    (a) `goal` is reached or
    (b) m-line is re-encountered at Q such that $Q \neq H_i$, $d(Q, \texttt{goal}) < d(H_i, \texttt{goal})$, and line $(Q, \texttt{goal})$ does not cross the current obstacle at Q or
    (c) H_i is re-encountered
  if `goal` is reached **then** exit with success

Bug2 Algorithm

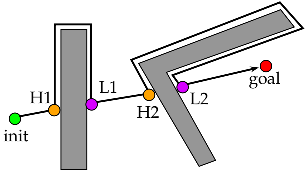
Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

call the line from `init` to `goal` the *m*-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the *m*-line again
  - leave the obstacle and continue straight toward `goal`

Bug2 Pseudocode



$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
  repeat move on a straight line from L_{i-1} to `goal`
  until `goal` is reached or obstacle is encountered at H_i
  if `goal` is reached **then** exit with success
  repeat follow boundary
  until
    (a) `goal` is reached or
    (b) *m*-line is re-encountered at Q such that $Q \neq H_i$, $d(Q, \texttt{goal}) < d(H_i, \texttt{goal})$, and line $(Q, \texttt{goal})$ does not cross the current obstacle at Q or
    (c) H_i is re-encountered
  if `goal` is reached **then** exit with success
  else if H_i is re-encountered **then** exit with failure

Bug2 Algorithm

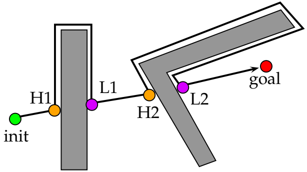
Vladimir J. Lumelsky and Alexander A. Stepanov: Algorithmica (1987) 2:403–430

call the line from `init` to `goal` the *m*-line

repeat until `goal` is reached

- head toward `goal`
- if sensor reports contact with an obstacle then
  - follow the obstacle until it encounters the *m*-line again
  - leave the obstacle and continue straight toward `goal`

Bug2 Pseudocode



$L_0 \leftarrow$ `init`; $i \leftarrow 1$
loop
    repeat move on a straight line from L_{i-1} to `goal`
    until `goal` is reached or obstacle is encountered at H_i
    if `goal` is reached **then** exit with success
    repeat follow boundary
    until
    (a) `goal` is reached or
    (b) *m*-line is re-encountered at Q such that $Q \neq H_i$, $d(Q, \texttt{goal}) < d(H_i, \texttt{goal})$, and line $(Q, \texttt{goal})$ does not cross the current obstacle at Q or
    (c) H_i is re-encountered
    if `goal` is reached **then** exit with success
    else if H_i is re-encountered **then** exit with failure
    else $L_i \leftarrow Q$; $i \leftarrow i + 1$

Lemma 3: Bug2 meets only a finite number of obstacles. Moreover, the only obstacles that can be met are those that intersect the straight-line segment $(init, goal)$

Bug2 Analysis

Lemma 3: Bug2 meets only a finite number of obstacles. Moreover, the only obstacles that can be met are those that intersect the straight-line segment $(init, goal)$

Lemma 4: Bug2 will pass any point of the i-th obstacle boundary at most $n_i/2$ *times, where* n_i *is the number of intersections between the straight line* $(init, goal)$ *and the i-th obstacle*

Bug2 Analysis

Lemma 3: $\mathit{Bug2}$ meets only a finite number of obstacles. Moreover, the only obstacles that can be met are those that intersect the straight-line segment $(\mathit{init}, \mathit{goal})$

Lemma 4: $\mathit{Bug2}$ will pass any point of the i-th obstacle boundary at most $n_i/2$ times, where n_i is the number of intersections between the straight line $(\mathit{init}, \mathit{goal})$ *and the i-th obstacle*

Theorem: $\mathit{Bug2}$ is a complete path-planning algorithm. Moreover, the length of a path generated by $\mathit{Bug2}$ never exceeds the limit

$$d(\mathit{init}, \mathit{goal}) + \sum_i \frac{n_i p_i}{2},$$

where p_i's refer to the perimeters of the obstacles intersecting the straight-line segment $(\mathit{init}, \mathit{goal})$

Bug2 Analysis

Lemma 3: Bug2 *meets only a finite number of obstacles. Moreover, the only obstacles that can be met are those that intersect the straight-line segment* $(init, goal)$

Lemma 4: Bug2 *will pass any point of the i-th obstacle boundary at most* $n_i/2$ *times, where* n_i *is the number of intersections between the straight line* $(init, goal)$ *and the i-th obstacle*

Theorem: Bug2 *is a complete path-planning algorithm. Moreover, the length of a path generated by* Bug2 *never exceeds the limit*

$$d(init, goal) + \sum_i \frac{n_i p_i}{2},$$

where p_i*'s refer to the perimeters of the obstacles intersecting the straight-line segment* $(init, goal)$

Proof Sketch: Similar to proofs for Bug1. Proof of Lemma 4 is slightly different. Maybe an upcoming homework exercise?

Bug1 vs Bug2

Bug1 vs Bug2

Bug1 is an exhaustive search algorithm
– looks at all choices before commiting

Bug1 has a more stable performance

Bug2 is a greedy search algorithm
– takes first choice that looks better

Bug2 often outperforms Bug1, but not always

Bug1 is an exhaustive search algorithm
– looks at all choices before commiting

Bug1 has a more stable performance

Bug2 is a greedy search algorithm
– takes first choice that looks better

Bug2 often outperforms Bug1, but not always

Draw scenes in which Bug2 beats Bug1 and vice-versa

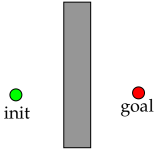
Bug1 is an exhaustive search algorithm
– looks at all choices before commiting

Bug1 has a more stable performance

Bug2 is a greedy search algorithm
– takes first choice that looks better

Bug2 often outperforms Bug1, but not always

Draw scenes in which Bug2 beats Bug1 and vice-versa



Bug2 beats Bug1

Bug1 vs Bug2

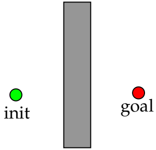
Bug1 is an exhaustive search algorithm
– looks at all choices before commiting

Bug1 has a more stable performance

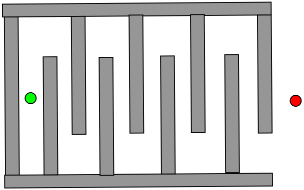
Bug2 is a greedy search algorithm
– takes first choice that looks better

Bug2 often outperforms Bug1, but not always

Draw scenes in which Bug2 beats Bug1 and vice-versa



Bug2 beats Bug1



Bug1 beats Bug2 in this scene

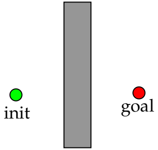
Bug1 is an exhaustive search algorithm
– looks at all choices before commiting

Bug1 has a more stable performance

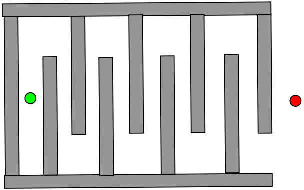
Bug2 is a greedy search algorithm
– takes first choice that looks better

Bug2 often outperforms Bug1, but not always

Draw scenes in which Bug2 beats Bug1 and vice-versa



Bug2 beats Bug1

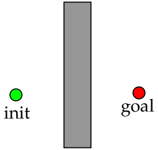


Bug1 beats Bug2 in this scene . . . but only if Bug2 always turns counterclock-wise or always turns clockwise when following boundary

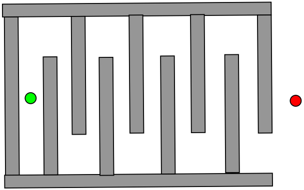
Bug1 is an exhaustive search algorithm
– looks at all choices before commiting
Bug1 has a more stable performance

Bug2 is a greedy search algorithm
– takes first choice that looks better
Bug2 often outperforms Bug1, but not always

Draw scenes in which Bug2 beats Bug1 and vice-versa



Bug2 beats Bug1



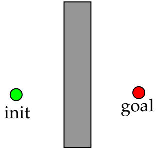
Bug1 beats Bug2 in this scene . . . but only if Bug2 always turns counterclock-wise or always turns clockwise when following boundary

what happens if Bug2 decides at random whether to turn counterclock-wise or clockwise each time it has follow an obstacle boundary?

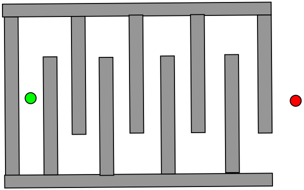
Bug1 is an exhaustive search algorithm
– looks at all choices before commiting
Bug1 has a more stable performance

Bug2 is a greedy search algorithm
– takes first choice that looks better
Bug2 often outperforms Bug1, but not always

Draw scenes in which Bug2 beats Bug1 and vice-versa



Bug2 beats Bug1



Bug1 beats Bug2 in this scene . . . but only if Bug2 always turns counterclock-wise or always turns clockwise when following boundary

what happens if Bug2 decides at random whether to turn counterclock-wise or clockwise each time it has follow an obstacle boundary?

can you draw a scene then where Bug1 beats Bug2 no matter how Bug2 decides to turn each time it has follow an obstacle boundary?

Raw Distance Function $\rho : \mathbb{R}^2 \times [0, 2\pi) \to \mathbb{R}$

$$\rho(x, \theta) = \min_{\alpha \in [0,\infty)} \alpha \text{ such that the point } x + \alpha \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \in \bigcup_i \texttt{Boundary}(O_i)$$

- $\rho(x, \theta)$ is the distance to the closest obstacle along the ray emanating from point $x \in \mathbb{R}^2$ at an angle $\theta \in [0, 2\pi)$

Saturated Raw Distance Function $\rho_R : \mathbb{R}^2 \times [0, 2\pi) \to \mathbb{R}$ with Sensing Range $R \in \mathbb{R}^{\geq 0}$

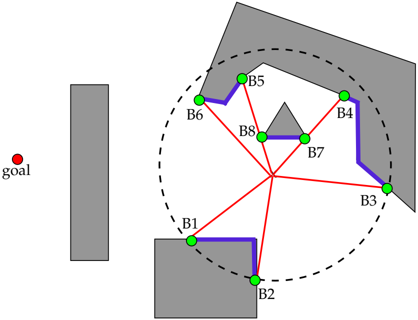
$$\rho_R(x, \theta) = \begin{cases} \rho(x, \theta), & \text{if } \rho(x, \theta) < R \\ \infty, & \text{otherwise} \end{cases}$$

- ρ_R has same value as ρ when obstacle is within sensing range R
- ρ_R has ∞ value when obstacles are outside the sensing range R

Ishay Kamon, Elon Romon, and Ehud Rivlin: IJRR (1998) 17:934–953

TangentBug relies on range sensor ρ_R to compute endpoints of finite continuous segments on obstacle boundaries

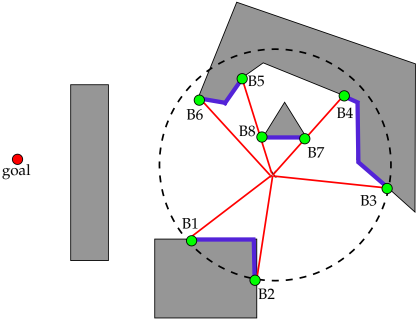
These segments constitute its local model of the world



Ishay Kamon, Elon Romon, and Ehud Rivlin: IJRR (1998) 17:934–953

`TangentBug` relies on range sensor ρ_R to compute endpoints of finite continuous segments on obstacle boundaries

These segments constitute its local model of the world

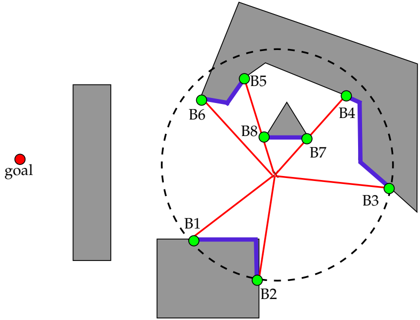


- `TangentBug` currently thinks it has unobstructed way to goal

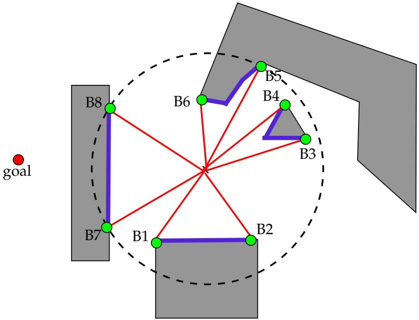
Ishay Kamon, Elon Romon, and Ehud Rivlin: IJRR (1998) 17:934–953

TangentBug relies on range sensor ρ_R to compute endpoints of finite continuous segments on obstacle boundaries

These segments constitute its local model of the world



- TangentBug currently thinks it has unobstructed way to goal

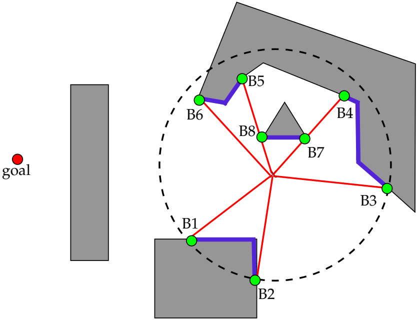


- TangentBug now sees that it can't go straight to the goal. What can it do?

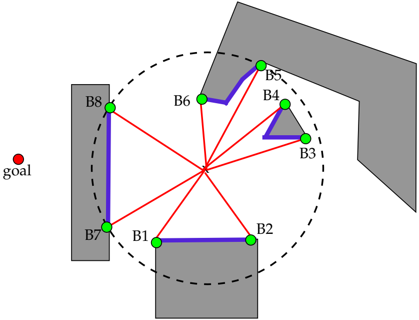
Ishay Kamon, Elon Romon, and Ehud Rivlin: IJRR (1998) 17:934–953

TangentBug relies on range sensor ρ_R to compute endpoints of finite continuous segments on obstacle boundaries

These segments constitute its local model of the world



- TangentBug currently thinks it has unobstructed way to goal

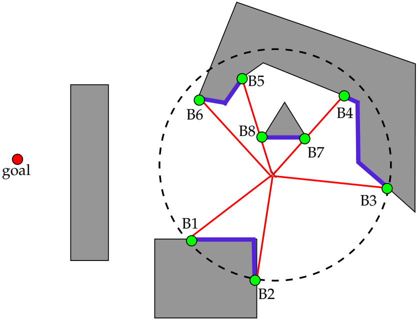


- TangentBug now sees that it can't go straight to the goal. What can it do?
- Choose the point B_i that minimizes heuristic distance $d(x, B_i) + d(B_i, \texttt{goal})$

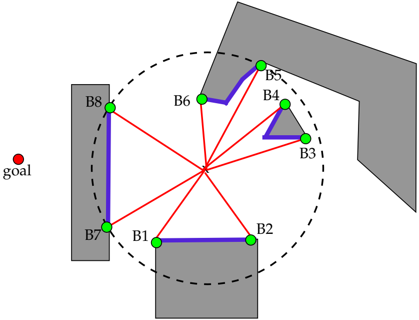
Ishay Kamon, Elon Romon, and Ehud Rivlin: IJRR (1998) 17:934–953

TangentBug relies on range sensor ρ_R to compute endpoints of finite continuous segments on obstacle boundaries

These segments constitute its local model of the world



- TangentBug currently thinks it has unobstructed way to goal

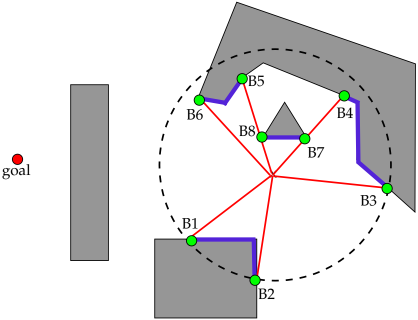


- TangentBug now sees that it can't go straight to the goal. What can it do?
- Choose the point B_i that minimizes heuristic distance $d(x, B_i) + d(B_i, \texttt{goal})$
- What if this distance starts increasing?

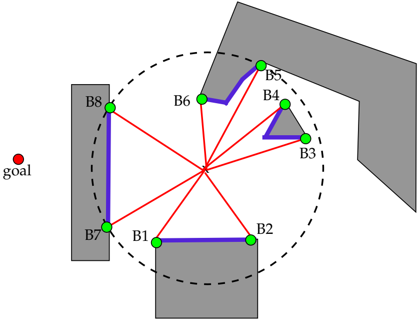
Ishay Kamon, Elon Romon, and Ehud Rivlin: IJRR (1998) 17:934–953

TangentBug relies on range sensor ρ_R to compute endpoints of finite continuous segments on obstacle boundaries

These segments constitute its local model of the world



- TangentBug currently thinks it has unobstructed way to goal



- TangentBug now sees that it can't go straight to the goal. What can it do?
- Choose the point B_i that minimizes heuristic distance $d(x, B_i) + d(B_i, \texttt{goal})$
- What if this distance starts increasing? Then, start following some boundary

TangentBug Algorithm – Basic Steps

- A motion-to-goal behavior as long as way is clear or there is a visible obstacle boundary point that decreases heuristic distance
- A boundary following behavior invoked when heuristic distance increases
- A value $d_{followed}$ which is the shortest distance between the sensed boundary and goal
- A value d_{reach} which is the shortest distance between blocking obstacle and goal (or distance to goal if no blocking obstacle visible)
- Terminate boundary following behavior when $d_{reach} < d_{followed}$

TangentBug Algorithm – Pseudocode

repeat until `goal` is reached

1. repeat
   - take sensor-range reading and compute continuous range segments
   - move toward point $n \in \{\texttt{goal}, B_1, B_2, \ldots\}$ that minimizes $h(x, n) = d(x, n) + d(n, \texttt{goal})$

   until
   - goal is reached, or
   - value of $h(x, n)$ begins to increase
2. follow boundary continuing in same direction as before repeating
   - update discontinuity points $\{B_1, B_2, \ldots\}$, d_{reach}, $d_{followed}$

   until
   - goal is reached, or
   - a complete cycle is performed (goal is unreachable)
   - $d_{reach} < d_{followed}$

Completeness proof similar to other bug-algorithm proofs, although the definition of hit and leave points is trickier

Basic problem: compute tangent to curve forming boundary of obstacle at any point, and drive the robot in that direction

- Let $D(x) = \min_c d(x, c), c \in \bigcup \texttt{Boundary}(O_i)$
- Let $G(x) = D(x) - W$, where W is some safe following distance
- Note that $\nabla G(x)$ points radially away from the object
- Define $T(x) = (\nabla G(x))$ the tangent direction
  - in a real sensor, this is just the tangent to the array element with lowest reading
- We could just move in the direction $T(x)$
  - open-loop control

Summary

- `Bug0` is incomplete
- `Bug1` is complete, safe, and reliable
- `Bug2` is complete, better in some cases than `Bug1`, but worse in others
- `TangentBug` is complete, supports range sensors

Reactive paradigm with minimal global information

Point Robot, Simple Motions

- Move straight toward `goal`
- Move along obstacle boundary
- Stop