Robot Exploration with Fast Frontier Detection: Theory and Experiments

Matan Keidar Gal A. Kaminka

MAVERICK Group, Department of Computer Science
Bar-Ilan University

AAMAS 2012

Outline

- Frontier-Based Exploration: a fundmental problem in robotics
- Variety of applications:
  - Search and Rescue [Kitano et al., 1999]
  - Planetary Exploration [Apostolopoulos et al., 2001]
  - Surveillance [Hougen et al., 2000]
- Previous methods for computing frontiers: Slow!
- Wavefront Frontier Detector (*WFD*)
- Fast Frontier Detector (*FFD*)
- Results: improvement of 1-2 orders of magnitude over *SOTA*

Frontier-Based Exploration

- The most common approach to exploration is based on *frontiers*
- *Frontier*: separates known regions from unknown regions
  - Set of *unknown* points
  - Each has at least one *open-space* neighbor
- By moving towards frontiers, robots keep discovering new regions
- Yamauchi was the first to show a frontier-based strategy
  - [Yamauchi, 1997, 1998]
- His work preceeded many others
  - [Burgard et al., 2005, Lau and NSW, 2003, Sawhney et al., 2009]

Frontier-Based Exploration

- The most common approach to exploration is based on *frontiers*
- *Frontier*: separates known regions from unknown regions
  - Set of *unknown* points
  - Each has at least one *open-space* neighbor
- By moving towards frontiers, robots keep discovering new regions
- Yamauchi was the first to show a frontier-based strategy
  - [Yamauchi, 1997, 1998]
- His work preceeded many others
  - [Burgard et al., 2005, Lau and NSW, 2003, Sawhney et al., 2009]

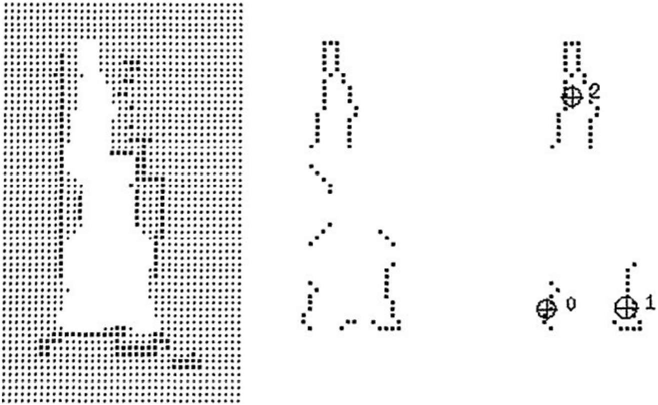


Figure: Image taken from [Yamauchi, 1998]: evidence grid, frontier points, extraction of different frontiers (from left to right).

Existing Frontier Detection is Slow

- Frontier detection algorithms rely on computer vision methods
  - e.g. edge detection and region extraction
- They have to process the entire map data with every execution
- Existing frontier detection methods take ~10–30 seconds to run
  - Even on powerful computers
  - Exploring a large area forces the robot to wait in its spot
  - Frontier detection is called only when the robot arrives at its target

Result

Real-time frontier detection can shorten the exploration time

Existing Frontier Detection is Slow

- Frontier detection algorithms rely on computer vision methods
  - e.g. edge detection and region extraction
- They have to process the entire map data with every execution
- Existing frontier detection methods take $\sim$10–30 seconds to run
  - Even on powerful computers
  - Exploring a large area forces the robot to wait in its spot
  - Frontier detection is called only when the robot arrives at its target

Result

Real-time frontier detection can shorten the exploration time

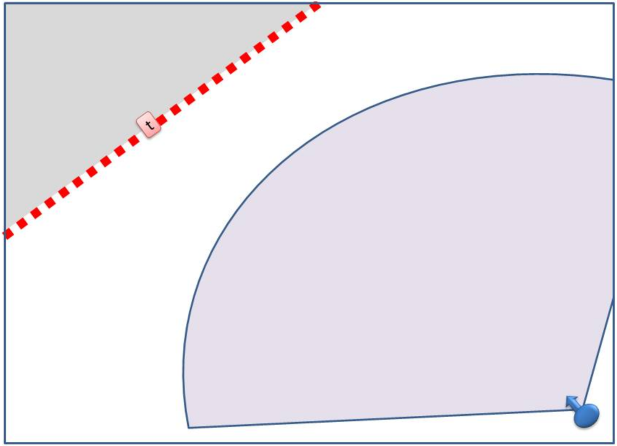
Existing Frontier Detection is Slow

- Frontier detection algorithms rely on computer vision methods
  - e.g. edge detection and region extraction
- They have to process the entire map data with every execution
- Existing frontier detection methods take $\sim$10–30 seconds to run
  - Even on powerful computers
  - Exploring a large area forces the robot to wait in its spot
  - Frontier detection is called only when the robot arrives at its target

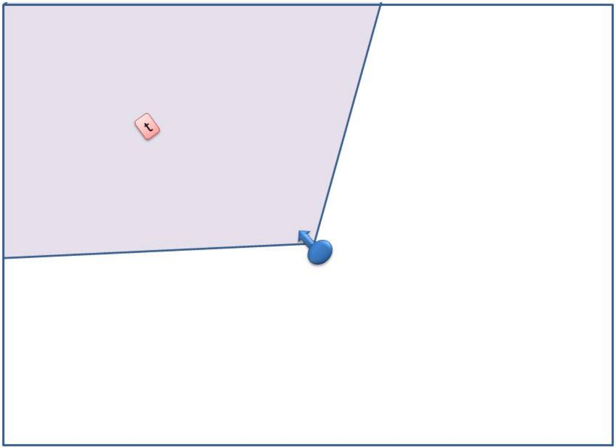
Result

Real-time frontier detection can shorten the exploration time

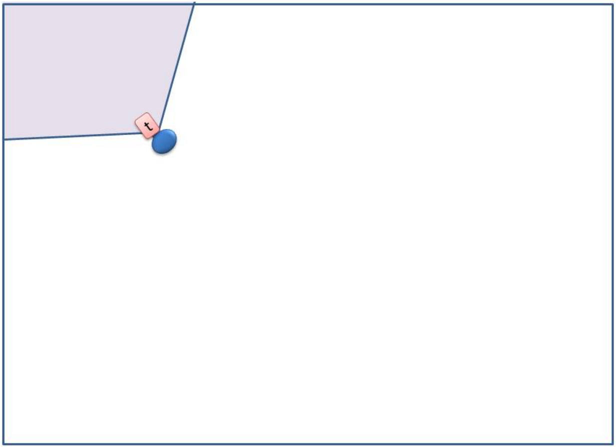
Single-Robot Example



Single-Robot Example



Single-Robot Example



WFD: Wavefront Frontier Detector

- Breadth-first search approach for frontier detection
- *WFD* avoids searching unknown regions
- *WFD* scans only known regions

WFD Outline

1. Enqueue current robot position
2. Perform Breadth-First Search
   - scan only open-space points that were not previously scanned
3. For every dequeued point, check if it is a frontier point
   - If True: extract the frontier by using another Breadth-First Search

WFD Conclusions

- Ensures that only known regions are actually scanned
- Frontier points are adjacent to open space points
  - All relevant frontiers will be found when *WFD* finishes
  - Connectivity of frontier points ensures complete frontier extraction
  - The algorithm does not have to scan the entire grid each time
- *WFD* still searches frontiers in all known space

FFD: Fast Frontier Detector

- Insights:
  - New frontiers are never contained within known regions
  - New frontiers are never wholly within unknown regions
- Hence, scanning all known regions is definitely unnecessary
  - and not time-efficient
- *FFD* avoids searching both known and unknown regions
  - Only processes new laser readings

FFD: Fast Frontier Detector

- Insights:
  - New frontiers are never contained within known regions
  - New frontiers are never wholly within unknown regions
- Hence, scanning all known regions is definitely unnecessary
  - and not time-efficient
- *FFD* avoids searching both known and unknown regions
  - Only processes new laser readings

FFD: Fast Frontier Detector

- Insights:
  - New frontiers are never contained within known regions
  - New frontiers are never wholly within unknown regions
- Hence, scanning all known regions is definitely unnecessary
  - and not time-efficient
- *FFD* avoids searching both known and unknown regions
  - Only processes new laser readings

FFD Outline

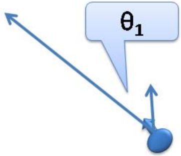
1. Sorting
2. Contour
3. Detecting New Frontiers
4. Maintaining Previously Detected Frontiers

Sorting

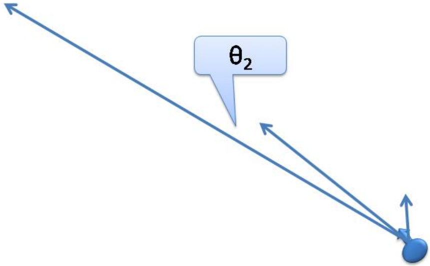
- Most laser sensors return readings that are already sorted
  - Points that are sorted according to polar angle
  - The robot as center
- However, if this is not the case, we can sort them efficiently

Sorting in details

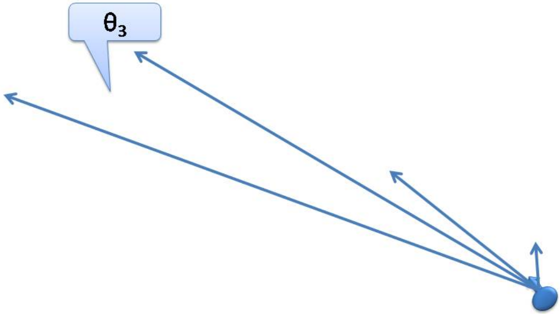
Sorting



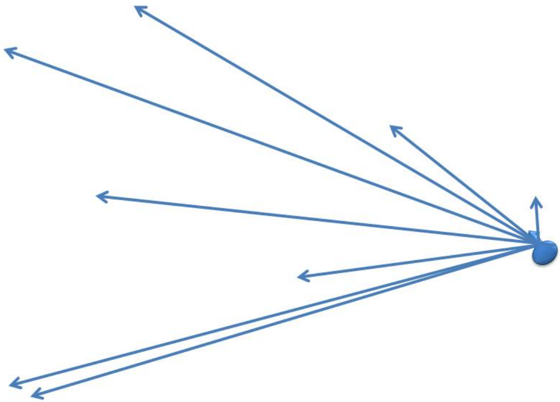
Sorting



Sorting



Sorting



Contour

- **Input**: sorted set of points
- **Output**: a contour that is built from the laser readings set
    - The line that connects each two adjacent points from the set
- Calculate the points that lie between each adjacent laser readings
- The desired contour contains all the points mentioned above

Contour

Problem

We need the algorithm to be fast and robust against rounding errors

Solution

We use *Bresenham's line algorithm* [Bresenham, 1965]

Contour

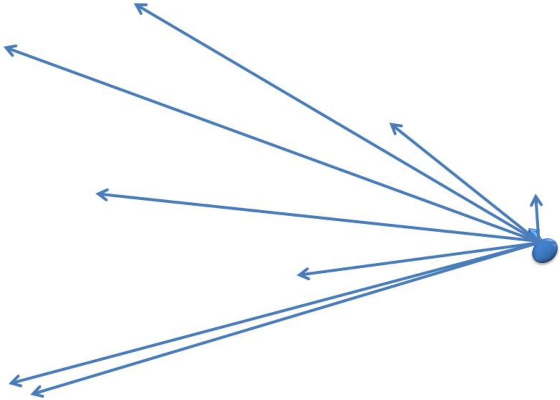
Problem

We need the algorithm to be fast and robust against rounding errors

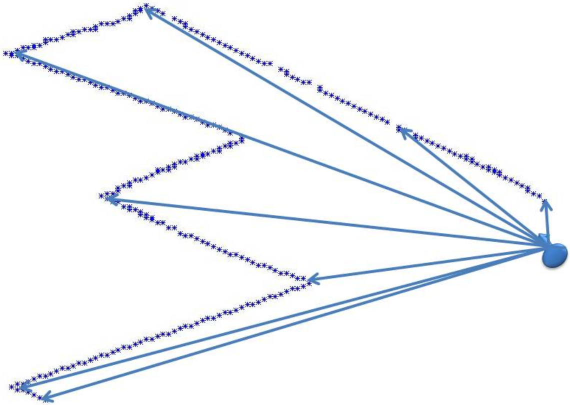
Solution

We use *Bresenham's line algorithm* [Bresenham, 1965]

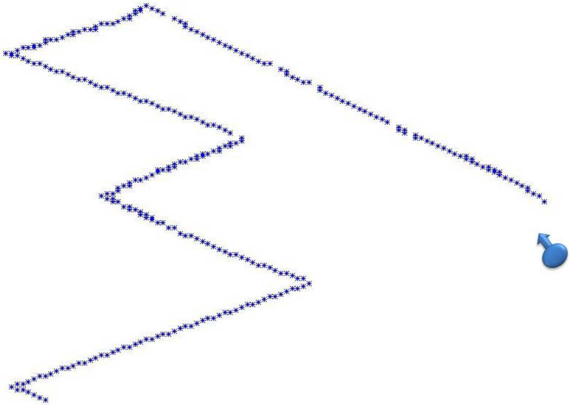
Contour



Contour



Contour



Detecting New Frontiers

- We scan the calculated contour from previous step
- Each point is compared with its (already scanned) predecessor
- Four possible cases:
  - Scanned point is not a frontier cell
  - Scanned point is a frontier cell but its predecessor is not
  - Both scanned point and its predecessor are frontier points
  - Scanned point is not a frontier cell but its predecessor is
- Full details can be found in the paper

Detecting New Frontiers

- We scan the calculated contour from previous step
- Each point is compared with its (already scanned) predecessor
- Four possible cases:
  - Scanned point is not a frontier cell
  - Scanned point is a frontier cell but its predecessor is not
  - Both scanned point and its predecessor are frontier points
  - Scanned point is not a frontier cell but its predecessor is
- Full details can be found in the paper

Maintenance: Motivation

- *FFD* gains its speed by only processing the laser readings
  - rather than entire regions of the map
- Previously detected frontiers are not updated during navigation
- Only the frontier that the robot is headed to

Problem

FFD is able to detect only *new* frontiers in each execution

Solution

Maintenance of frontiers which are not covered in the sensors range

Maintenance: Motivation

- *FFD* gains its speed by only processing the laser readings
  - rather than entire regions of the map
- Previously detected frontiers are not updated during navigation
- Only the frontier that the robot is headed to

Problem

FFD is able to detect only *new* frontiers in each execution

Solution

Maintenance of frontiers which are not covered in the sensors range

Maintenance 1: Eliminating Previously Detected Frontiers

- Points which are no longer in frontiers have to be eliminated
- *Active Area*: blocking rectangle constructed from laser readings
- If a frontier is to be eliminated, it must lie inside the active area
  - it contains regions that are covered by the robot's sensors
- *FFD* scans each point that lies inside the *Active Area*
  - Checking if a point is previously belonged to a frontier is fast
  - Full details can be found in the paper

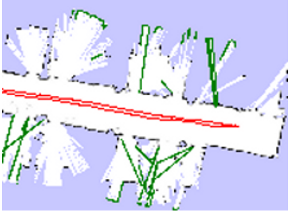
Maintenance 2: Avoiding Re-detection of Same Frontier

Problem

FFD might wrongly detect a new frontier in an already scanned area

Solution

FFD distinguishes between laser readings in different time frames



Full details can be found in the paper

Maintenance: Summary

- *FFD* has to run in the background
    - In contrast to other approaches that are executed in a certain time
- *FFD* requires robustness against map orientation changes
    - caused by loop-closures

Experiment Design

- We have fully implemented *WFD* and *FFD*
- We compare *WFD* and *FFD* with a State-of-the-Art algorithm[1]
- Our system is based on *GMapping* SLAM implementation
  - [Grisetti et al., 2005, 2007]
- Two machines:
  - Intel Core 2 Duo T6600 CPU, 2.20GHz, 4GB RAM
  - Intel Pentium III Coppermine CPU, 800MHz, 1GB RAM
- We measured CPU-process time

[1] We thank Kai M. Wurm and Wolfram Burgard for providing us their own implementation

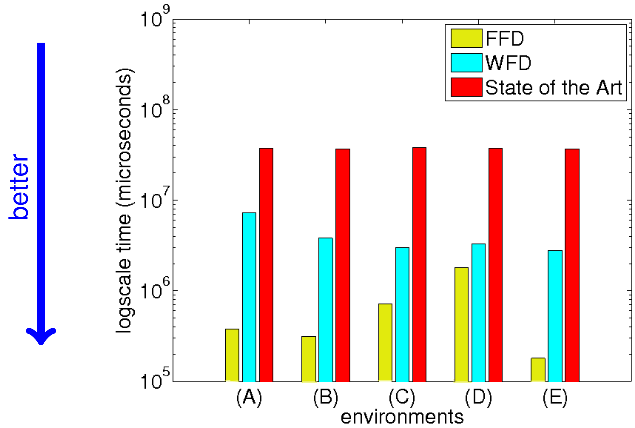
Experiment Design

- All detection algorithms are executed when a map update occurs
- *FFD* is executed when a new laser reading is received
- We accumulate *FFD*'s time between calls to other algorithms
- Tested on data obtained from RADISH [Howard and Roy, 2003]

Experiment Design

Figure: Example of a testing environment, University of Freiburg. Image was taken from RADISH, Howard and Roy [2003]

Results



- Y axis: mean execution time (microseconds) on a *logaritmic scale*
- *WFD* is faster than *SOTA* by two orders of magnitude
- *FFD* is faster than *WFD* by an order of magnitude

Summary and Future Work

- Real-Time frontier detection on weak CPU's
- *WFD*, *FFD*: two methods for frontier detection
  - significant reduction of calculation time
- Future work:
  - Address efficient methods for maintaining frontiers in *FFD*
  - Novel exploration policies based on real-time frontier-detection
- For more information:
  - `matankdr@gmail.com`
  - `galk@cs.biu.ac.il`

Sorting

- The first step sorts laser readings based on their angle
  - i.e based on the polar coordinates with the robot as the origin
- The naive method for converting Cartesian coordinates to polar coordintates is time-consuming
  - calling *atan2* and *sqrt*
- By using *Cross-Product* we can perform sorting much faster
  - $(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0) \cdot (y_2 - y_0) - (x_2 - x_0) \cdot (y_1 - y_0)$
  - If the result is positive, then $\overrightarrow{P_0 P_1}$ is clockwise from $\overrightarrow{P_0 P_2}$.
  - Else, it is counter-clockwise.
  - If result is 0, then the two vectors lie on the same line in the plane

Back to *FFD*

Sorting

- The first step sorts laser readings based on their angle
  - i.e based on the polar coordinates with the robot as the origin
- The naive method for converting Cartesian coordinates to polar coordintates is time-consuming
  - calling *atan2* and *sqrt*
- By using *Cross-Product* we can perform sorting much faster
  - $(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0) \cdot (y_2 - y_0) - (x_2 - x_0) \cdot (y_1 - y_0)$
  - If the result is positive, then $\overrightarrow{P_0 P_1}$ is clockwise from $\overrightarrow{P_0 P_2}$.
  - Else, it is counter-clockwise.
  - If result is 0, then the two vectors lie on the same line in the plane

Back to *FFD*

maverick

Bar-Ilan University
אוניברסיטת בר-אילן

Sorting

- The first step sorts laser readings based on their angle
  - i.e based on the polar coordinates with the robot as the origin
- The naive method for converting Cartesian coordinates to polar coordintates is time-consuming
  - calling *atan2* and *sqrt*
- By using *Cross-Product* we can perform sorting much faster
  - $(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0) \cdot (y_2 - y_0) - (x_2 - x_0) \cdot (y_1 - y_0)$
  - If the result is positive, then $\overrightarrow{P_0 P_1}$ is clockwise from $\overrightarrow{P_0 P_2}$.
  - Else, it is counter-clockwise.
  - If result is 0, then the two vectors lie on the same line in the plane

Back to *FFD*

Sorting

- The first step sorts laser readings based on their angle
  - i.e based on the polar coordinates with the robot as the origin
- The naive method for converting Cartesian coordinates to polar coordintates is time-consuming
  - calling *atan2* and *sqrt*
- By using *Cross-Product* we can perform sorting much faster
  - $(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0) \cdot (y_2 - y_0) - (x_2 - x_0) \cdot (y_1 - y_0)$
  - If the result is positive, then $\overrightarrow{P_0 P_1}$ is clockwise from $\overrightarrow{P_0 P_2}$.
  - Else, it is counter-clockwise.
  - If result is 0, then the two vectors lie on the same line in the plane

Back to *FFD*

D.S. Apostolopoulos, L. Pedersen, B.N. Shamah, K. Shillcutt, M.D. Wagner, and W.L. Whittaker. Robotic antarctic meteorite search: Outcomes. In *IEEE International Conference on Robotics and Automation*, pages 4174–4179, 2001.

W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3): 376–378, 2005.

G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2443–2448, 2005.

G. Grisetti, C. Stachniss, and W Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23:34–46, 2007.

Dean F. Hougen, Saifallah Benjaafar, Jordan Bonney, John Budenske, Mark Dvorak, Maria L. Gini, Howard French, Donald G. Krantz, Perry Y. Li, Fred Malver, Bradley J. Nelson, Nikolaos Papanikolopoulos, Paul E. Rybski, Sascha Stoeter, Richard M. Voyles, and Kemal Berk Yesin. A miniature robotic system for reconnaissance and surveillance. In *ICRA*, pages 501–507, 2000.

Andrew Howard and Nicholas Roy. The robotics data set repository (RADISH), 2003. URL `http://radish.sourceforge.net/`.

Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsuhi Shinjou, and Susumu Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 739–746. IEEE Computer Society, 1999.

H. Lau and A. NSW. Behavioural approach for multi-robot exploration. In *Australasian Conference on Robotics and Automation (ACRA), Brisbane, December*, 2003.

R. Sawhney, K. M Krishna, and K. Srinathan. On fast exploration in 2D and 3D terrains with multiple robots. In *Proceedings of the 8th*

International Conference on Autonomous Agents and Multiagent Systems. Vol. 1, pages 73–80, 2009.

B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8138-1.

B. Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 47–53, 1998.