Design of Biologically Inspired Topological Maps for Robot Navigation

Marcus Augustine, M. Sc.
1. Motivation
2. Goal
3. Approach
4. Results
5. Outlook
Navigation and Mapping
- Latest Solutions (SLAM) – geometric consistent maps of the environment
- Performance decreases with growing map size, suitable only for bounded environments
- Problem: long-range, robust & efficient navigation in mobile robotics
- Biology: Insects – foraging behavior
- Limited storage and processing power
• Algorithm for building a topological map in 2D/ 3D
• Scaling, efficient storage; adaption to current situation, limited storage
• Based on landmarks, arranged in a graph
• Metrical information omitted (range)
• Robot movement based on (noisy) landmark bearings
• Sensor type for bearing measurements not specified
• Similar to insects: known path should be traveled forwards and backwards
Motivation • Goal • Approach • Results • Outlook

Metrical vs. Topological Maps

Real Environment

Metric Map

Topological Map
• Needs unique landmarks and their bearings
• Compass information
• Concentrates on translation
Insects: Navigation based on landmarks/panorama
Localisation: retinoptical ability
Snapshot model by Cartwright and Collett

Components
- Representation of distinguishable places (Viewframes)
- Local homing (Viewframe Homing)
- Numerical similarity measure (Viewframe Classifier)
- Data-structure including places (Topological Map)
• 3 Methods
  • Tangential correction vectors
  • Secant correction vectors
  • Closed-form solution based on Lagrange-Multipliers
- Mean Square Error (MSE) Measure

\[ \delta_{\text{MSE}} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} (a_{ij} - b_{ij})^2}{nm} \]

- Pseudo-Huber Cost-Function-based Classifier

\[ \delta_{\text{Pseudo-Huber}} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} 2c^2 \left( \sqrt{1 + \left( \frac{a_{ij} - b_{ij}}{c} \right)^2} - 1 \right)}{nm} \]
Motivation • Goal • Approach • Results • Outlook
• Modular simulation- and visualisation-environment in MATLAB (C++ API for external routines)
  • Measurement aberrations
  • Occlusions, outliers
• Data-structure in C++ (STL-types & -algorithms)
Results – Homing

Motivation • Goal • Approach • Results • Outlook

- strong noise: angular measurement deviation ($\sigma_{\theta_{\text{meas}}} = 5.0$), no occlusions, no outliers.
- outliers: no measurement deviation of the landmark measurement, no occlusions, 5% outliers ($P_{\text{occ}} = 0.05$).
- occlusions: angular measurement deviation ($\sigma_{\theta_{\text{meas}}} = 0.1$), 10% occlusions ($P_{\text{occ}} = 0.1$), no outliers.

(a) ideal conditions.
(b) strong noise.
(c) outliers.
(d) occlusions.
Results – Tree Pruning (1/2)

Motivation • Goal • Approach • Results • Outlook

Tree height 9
Data 93%
39 viewframes

Tree height 8
Data 88%
37 viewframes

Tree height 7
Data 83%
35 viewframes

Tree height 6
Data 71%
31 viewframes

Tree height 5
Data 60%
27 viewframes

Tree height 4
Data 45%
22 viewframes
Results – Tree Pruning (2/2)

- **Motivation**
- **Goal**
- **Approach**
- **Results**
- **Outlook**

**Results**

**Tree height 3**
- Data 28%
- 16 viewframes

**Tree height 2**
- Data 17%
- 11 viewframes

**Tree height 1**
- Data <3%
- 6 viewframes
• Successful realisation of a simulation- and visualisation environment in MATLAB
• Homing applicable in the presence of noise
• MSE sensible towards outliers, occlusions and angular aberrations
  • Pseudo-Huber-based Classifier supresses this behavior
• Comparison and Benchmarking of the homing methods
• Data-structure suitable for indoor/ outdoor use
• Maximum reduction expected in outdoor environment
• Improved classifiers (different cost functions)
• Adaptive threshold value
• Landmark database (metainformation for more robust identification)
• Approach on real platform, ICRA 2012 paper (Sep 2012)
Thanks for your attention
Backup
Motivation

Goal

Approach

Results

Outlook

\[ \vec{H} = \sum_{n=1}^{n} \vec{v}_i \]

\[ \vec{v}_i = M \cdot \hat{c}_i \]

\[ M = \frac{\arctan(\hat{c}_{iy}, \hat{c}_{iz}) - \arctan(\hat{s}_{iy}, \hat{s}_{iz})}{\pi} \]

\[ R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \]

\[ \hat{c}_i = \begin{cases} R(\frac{\pi}{2}) \cdot \hat{c}_i & \text{if } \angle \hat{c}_i \leq \angle \hat{s}_i \\ R(-\frac{\pi}{2}) \cdot \hat{c}_i & \text{else} \end{cases} \]
\[ \vec{H} = \sum_{n=1}^{n} \vec{v}_i \]  
(7.4.6)

\[ \vec{v}_i = \vec{e}_i - \vec{s}_i \]  
(7.4.7)
Motivation

Goal

Approach

Results

Outlook

\[ E(t, \lambda) = \sum_i w_i (n_i^T t)^2 - \lambda(t^2 - 1), \]
\[ = \sum_i w_i n_i^T n_i t - \lambda(t^2 - 1), \]
\[ = t^T (\sum_i w_i n_i n_i^T) t - \lambda(t^2 - 1) \]  
\[ 0 = \sum_i w_i (n_i^T t)n_i - 2\lambda t, \]
\[ \lambda t = \sum_i w_i (n_i^T t)n_i \]
\[ = \sum_i w_i n_i (n_i^T t) \]  
\[ \lambda t = (\sum_i w_i n_i (n_i^T t) \]
\[ \hat{N} = \sum_i w_i n_i n_i^T \]
\[ = \begin{pmatrix}
\sum_i w_i n_{zi}^2 & \sum_i w_i n_{zi}n_{yi} & \sum_i w_i n_{zi}n_{zi} \\
\sum_i w_i n_{zi}n_{yi} & \sum_i w_i n_{yi}^2 & \sum_i w_i n_{yi}n_{zi} \\
\sum_i w_i n_{zi}n_{zi} & \sum_i w_i n_{yi}n_{zi} & \sum_i w_i n_{zi}^2
\end{pmatrix} \]
\[ d = \sum_i e_i - e_i' \]
\[ \hat{R} \begin{cases} 
\hat{R}' & \arccos \left( \frac{d \cdot \hat{R}'}{|d| \cdot |\hat{R}'|} \right) < \frac{\pi}{2} \\
-\hat{R}' & \text{else}
\end{cases} \]
• Construction of the tree
  • TransformViewframeListToTree
  • AppendViewframeToTree
• Manipulation and Administration
  • UpdateNodeHeight
  • TruncateTreeAtLevel
  • GetViewframe
TransformViewframeListToTree

Algorithm 1: TransformViewframeListToTree

1. input: a list of viewframes $List_V$
2. output: a tree $T$ representation of the viewframes in $List_V$
3. precondition: $List_V$ is not empty, $T$ is empty
4. postcondition: $|T| = |List_V|$, $T$ is valid tree
5. $T \leftarrow \text{empty}$
6. while $List_V$.hasNextViewframe do
7. \hspace{1cm} $V_{\text{current}} \leftarrow List_V$.nextViewframe
8. \hspace{1cm} $T = \text{AppendViewframeToTree}(V_{\text{current}}, T)$
9. return $T$

Complexity $O(m \cdot O(\text{AppendViewframeToTree}))$
**Motivation**

**Goal**

**Approach**

**Results**

**Outlook**

---

**Algorithm 2: AppendViewframeToTree**

```plaintext
input: a viewframe V, a tree T
output: a tree T'  
postcondition: T' ≠ T

1. if T' is not null then
   2. T'.root = create node that includes Ø
   3. Nodeprev = T'.root
   4. while T'.hasElements do
      5. (SameElements) = collect same elements in Nodeprev, latestChild and V
      6. (RemainingElementsViewframe) = Nodeprev, latestChild.elements \ (SameElements)
      7. (RemainingElements) = V.elements \ (SameElements)
      8. if (SameElements) is @ then
         9. TemporaryNode = create node that includes V.elements
         10. Nodeprev.Children = TemporaryNode
         11. UpdateNodeHeight of TemporaryNode
         12. V.elements = Ø
      13. else if Nodeprev, latestChild.elements = V.elements then
         14. if Nodeprev, latestChild hasChildren then
            15. Nodeprev, latestChild.V.elements = (RemainingElementsViewframe)
      16. else
         17. TemporaryNode = create node that includes (RemainingElementsViewframe)
         18. Nodeprev, latestChild.Children = TemporaryNode
         19. UpdateNodeHeight of TemporaryNode
         20. V.elements = Ø
      21. end if
   4. else
      5. TemporaryNode = create node that includes (RemainingElementsViewframe)
      7. TemporaryNode.height = Nodeprev, latestChild, height
      8. Nodeprev, latestChild, deprecateChild, Nodeprev, latestChild, Children
      9. Nodeprev, latestChild, Children = TemporaryNode
      10. (RemainingElementsViewframe) = Nodeprev, latestChild, elements \ (SameElements)
      11. UpdateNodeHeight of TemporaryNode
      12. V.elements = Ø
   4. end if

return T'
```

**Complexity** $O(n) + O(UpdateNodeHeight)$

---

Slide 27  
Marcus Augustine  
06.07.12
AppendViewframeToTree
TransformViewframeListToTree • AppendViewframeToTree • UpdateNodeHeight • TruncateTreeAtLevel • GetViewframe
Motivation • Goal • Approach • Results • Outlook

UpdateNodeHeight

TransformViewframeToListToTree • AppendViewframeToTree • UpdateNodeHeight • TruncateTreeAtLevel • GetViewframe

Algorithm 3: UpdateNodeHeight

\[
\text{input} : \text{a node } N_p, \text{ a tree } T \\
\text{output} : \text{a tree } T' \\
\text{precondition} : T \text{ is not empty, might consist of nodes with incorrect height} \\
N_p \in T \text{ and has correct height} \\
\text{postcondition} : T' \text{ is not empty, all nodes have correct height}
\]

1. ProcessNode = \( N_p \)
2. Counter = ProcessNode.height
3. while ProcessNode.hasParent do
   4. \( \text{MaxHeightByNow} = 0 \)
   5. foreach TemporaryNode in ProcessNode.getParent.getChildren do
      6. if TemporaryNode.height < \( \text{MaxHeightByNow} \) then
         7. \( \text{MaxHeightByNow} = \text{TemporaryNode.height} \)
      8. if Counter < \( \text{MaxHeightByNow} \) then
         9. return \( T' \)
      10. else
          11. increase Counter
          13. ProcessNode.height = Counter
   14. return \( T' \)

Complexity \( \mathcal{O}(\max(m, n)) \)

Marcus Augustine
06.07.12
**Motivation**

- **Goal**
- **Approach**
- **Results**
- **Outlook**

TransformViewframeListToTree • AppendViewframeToTree • UpdateNodeHeight • TruncateTreeAtLevel • GetViewframe

---

**Algorithm 4: TruncateTreeAtLevel**

Input: a tree $T$, a level value $Level$
Output: a tree $T'$

Precondition: $T$ is not empty
$Level > 0$ and $Level \leq T$.height

Postcondition: $T'.height = Level - 1$

1. $Counter = 0$
2. initialize $ParentStack$
3. initialize $ChildrenStack$
4. $ParentStack \leftarrow T$.root
5. while $Counter < Level - 1$ do
6.   while $ParentStack$ is not empty do
7.     $ParentElement \leftarrow ParentStack$.top
8.     $ParentStack$.pop
9.     foreach $Child$ in $ParentElement$.getChildren do
10.    if $Child$.height + $Counter$ $\geq$ $Level$ - 1 then
11.      $ChildrenStack \leftarrow Child$
12.     Increase $Counter$
13.     swap $ParentStack$ and $ChildrenStack$  # $ParentStack$ is empty and $ChildrenStack$ contains new parents afterwards
14.     foreach $Parent$ in $ParentStack$ do
15.         $Parent$.releaseChildCustody
16.         $Parent$.height = 0
17.         UpdateNodeHeight of $Parent$
18.     return $T'$

Complexity $\mathcal{O}(\max(m, n))$
Algorithm 5: GetViewframe

- **input**: a leaf $N_p$, a tree $T$
- **output**: a set of (landmark id, landmark angle)-tuples representing a viewframe $V$
- **precondition**: $T$ is not empty, $N_p \in T$
- **postcondition**: $T$ is not empty, all nodes have correct height

1. $ProcessNode = N_p$
2. $ResultSet = \emptyset$
3. while $ProcessNode.hasParent$ do
   4. $ResultSet \leftarrow ProcessNode.elements$
   5. $ProcessNode = ProcessNode.parent$
4. return $ResultSet$

Complexity $O(n)$
Motivation • Goal • Approach • Results • Outlook

Snapshot Models

snapshot model  difference vector model  ALV model