Visual Programming for Mobile Robot Navigation Using High-level Landmarks

Joseph Lee[1], Yan Lu[2], Yiliang Xu[3], and Dezhen Song[1]

***Abstract*— We propose a visual programming system that allows users to specify navigation tasks for mobile robots using high-level landmarks in a virtual reality (VR) environment constructed from the output of visual simultaneous localization and mapping (vSLAM). The VR environment provides a Google Street View-like interface for users to familiarize themselves with the robot's working environment, specify high-level landmarks, and determine task-level motion commands related to each landmark. Our system builds a roadmap by using the pose graph from the vSLAM outputs. Based on the roadmap, the high-level landmarks, and task-level motion commands, our system generates an output path for the robot to accomplish the navigation task. We present data structures, architecture, interface, and algorithms for our system and show that, given n_s search-type motion commands, our system generates a path in $O(n_s(n_r \log n_r + m_r))$ time, where n_r and m_r are the number of roadmap nodes and edges, respectively. We have implemented our system and tested it on real world data.**

I. Introduction

As service-oriented robots increase, they have to be programmed by users without much technical background. The average user will need easier and more flexible robot programming tools as opposed to the specialized programming languages used for industrial robots [1]. For mobile robots, the most fundamental task is to program robot navigation capabilities, which direct a robot to move from point A to point B in an environment in which prior maps often are not available. For a novice user, generating a path of waypoints on a sparse landmark map produced by prior simultaneous localization and mapping (SLAM) outputs is nontrivial due to the combination of human's poor spacial reasoning capability and lack of contextual information.

Here we propose a visual programming system that allows users to specify navigation tasks for mobile robots using high-level landmarks in a virtual reality (VR) environment constructed from the output of visual SLAM (vSLAM) (See Fig. 1). The VR graphical user interface (GUI) of our system allows users to take a virtual tour similar to Google Street View to familiarize themselves with the robot's working environment. Users can directly specify image regions of objects as high-level landmarks from the scene. With each specified landmark, the user creates task-level motion commands. In the preprocessing step, our system builds a roadmap by using the pose graph from the vSLAM outputs. Based on the roadmap, the high-level landmarks, and task-level motion commands, our system generates an output path for the robot to accomplish the navigation task.

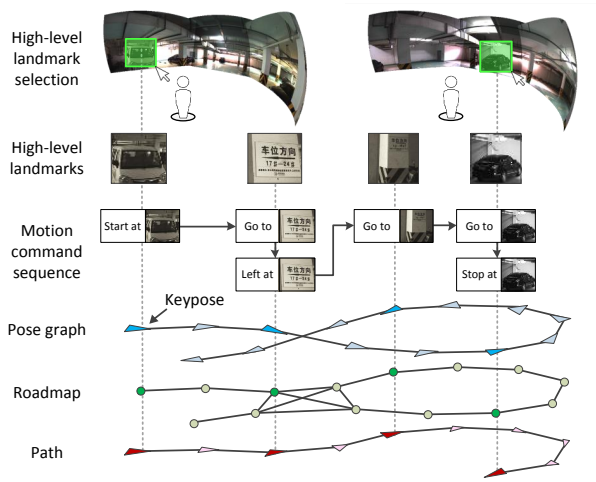

Fig. 1: Mobile robot programming with our proposed system (Best viewed in color). Our system generates a robot path from a user-defined motion command sequence that uses high-level landmarks. This enables users to program mobile robots at the object level without dealing with low-level map coordinates.

We present data structures, architecture, interface, and algorithms for our system. Given n_s search-type motion commands, our system generates a path in $O(n_s(n_r \log n_r + m_r))$ time, where n_r and m_r are the number of roadmap nodes and edges, respectively. We have implemented our system and tested it with real indoor data acquired by a mobile robot equipped with a synchronized camera array and a lidar sensor. The experimental results show that our system can generate robot paths that satisfy the user commands.

II. Related Work

Our proposed robot programming system relates to the areas of robot programming, motion planning, SLAM, and teleoperation.

In [1], Biggs and MacDonald provide a survey on robot programming which divides the methods into manual programming and automatic programming. In manual programming, the user directly programs the robot using either text-based [2] or graphical systems [3], whereas automatic programming refers to methods such as learning and programming by demonstration, a common technique used for manipulator robots [4], [5]. For mobile robots, Kanayama

This work was supported in part by National Science Foundation under IIS-1318638, NRI-1426752, and NRI-1526200, and in part by TxDot 0-6869.

[1]J. Lee and D. Song are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA. E-mail: {jslee,dzsong}@cse.tamu.edu

[2]Y. Lu is with Honda Research Institute, Mountain View, CA 94043, USA. E-mail: sinoluyan@gmail.com

[3]Y. Xu is with Apple Inc., Cupertino, CA 95014, USA. E-mail: yiliang_xu@ieee.org

and Wu [2] develop a high-level programming language for text-based programming. Nicolescu and Mataric [6] develop an instructive system that programs a mobile robot using learning by demonstration. However, the effort to train a mobile robot in this approach will grow linear to space, and many training examples are required to increase flexibility. Our system can be viewed as a new task-level automatic programming using VR.

Robot motions can be generated by motion planning algorithms with classic methods such as [7], [8]. However, motion planning requires a complete scene representation which is often unavailable for service mobile robots in unstructured environments. Also, motion planning can be a difficult task for a robot and can benefit from human input. For example, a human-assisted motion planner is proposed in [9] where the user can steer the planner towards/away from certain regions. In [10], a remote human-in-the-loop gripper is presented where the user can assist motion planning by moving a virtual gripper in the display and specifying waypoints. Our path generation step can be viewed as a motion planning problem with a given roadmap. However, inspired by the existing works, the main focus of our system is to translate user intention into the motion planning framework to automate the path generation process.

To display the robot's working environment, our system employs VR built by keyframes from vSLAM. We also use pose graph from the vSLAM as a starting point for the roadmap construction. SLAM can be performed using depth sensors and visual sensors, i.e. regular cameras. SLAM methods based on depth sensors such as lidar [11], [12] and RGB-D cameras [13], [14] have low scale-drift issues, whereas visual sensors provide both geometric and appearance data. In vSLAM, the extended Kalman filter [15] and bundle adjustment [16] are mainly used. In our work, we use a 2D lidar map generated by the method in [17] and use the pose graph and keyframes of the multi-layered feature graph (MFG) proposed by [18] which exploits the regularities of urban environment such as rectilinear structures.

Integrating trajectory following and obstacle avoidance capabilities [19], our system can be used as a supervisory control system in teleoperation. Fong and Thorpe [20] classify vehicle teleoperation interfaces into four categories: direct [21], multimodal/multisensor, supervisory control [22], and novel [23]. To allow a free-look of the environment, a visual teleoperation display is also proposed for unmanned vehicles using a spherical camera and an Oculus Rift [24]. It has been shown that VR can significantly improve telepresence. Our system falls into this category. However, directly controlling the robot can be difficult due to poor spacial reasoning of the user from the display. Our high-level landmark-based supervisory control can avoid this issue.

Our experience in vSLAM using mobile robots [18] and teleoperation [25], [26] has prompted a need for a system that allows users to easily assist robot navigation in urban environments.

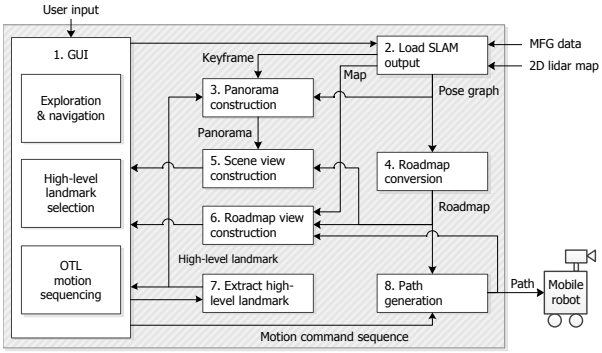

Fig. 2: System diagram

III. System Design

A. Overall structure

The system diagram of our visual programming system is depicted in Fig. 2. Using our system, a user explores and navigates through the robot's environment as a virtual tour and specifies high-level landmarks which are used for object-oriented task-level (OTL) motion sequencing. Given the OTL motion commands, our system generates a robot path.

From the GUI, the user first loads the MFG data and 2D lidar map to construct the robot's environment and roadmap. The environment is created using both keyframes and the pose graph of the MFG data, and the roadmap is created from the pose graph as a preprocessing step. The user navigates along the roadmap and keeps track of his current pose in the roadmap view, where the roadmap is overlaid on the 2D lidar map. From the scene view, the user selects high-level landmarks which are used for OTL motion sequencing. After the user completes motion sequencing, the system generates a path from the given motion commands. The robot path is then displayed to the user and used by a mobile robot.

The generated path is not limited to our robot but can be used among different type of mobile robots. When the path is generated off-line, the path can be used to program a mobile robot. However, as noted previously, our system can also be used as an interface for on-line robot control in a teleoperation scenario.

B. Data structures

The data structures in the MFG data that are used in our system are listed as follows:

- *Keyframe* is a camera image denoted by I_i^k, where k denotes the key index, and i denotes the camera index in the synchronized camera array.
- *Keypose* is an estimated robot pose when keyframe I_i^k is captured. Let the camera pose for I_i^k be defined by a rotation matrix $\mathrm{R}_i^k \in \mathrm{SO}(3)$ and a translation vector $\mathbf{t}_i^k \in \mathbb{R}^3$. Then the keypose is aligned with the reference camera pose in the camera array, i.e. R_0^k and $\mathbf{t}_0^k$.
- *Pose graph* is an undirected graph P, where nodes represent keyposes and edges represent their dependencies.

Additionally, the required data structures used to generate a robot path from user commands are illustrated in Fig. 1 and defined below.

- *High-level landmark* is a user-specified image region in I_i^k. The image region is defined by a bounding box and labeled by the user. However, if an image recognition algorithm is available, high-level landmarks can also be identified by the system.
- *Motion command* is a text command paired with a high-level landmark by the user. The text commands specify robot tasks, such as *start at*, *go to*, and *avoid*.
- *Motion command sequence* is a directed graph M, where a node m_i represents a motion command, and edge (m_i, m_j) represents the transition from m_i to a child node m_j. Each edge has a condition that should be satisfied to make the transition.
- *Roadmap* is an undirected graph R, where the k-th node r_k represents a waypoint $\mathbf{r}_k \in \mathbb{R}^3$. An edge (r_{k_1}, r_{k_2}) in R represents the Euclidean distance between r_{k_1} and r_{k_2}.
- *Path* is a continuous sequence S where the elements are robot positions in $\mathbb{R}^3$.

C. User Interface

In this section, we describe the GUI of our system (Fig. 2 Box 1) that facilitates robot programming using high-level landmarks. Our GUI, shown in Fig. 3, consists primarily of four components: *scene view*, *roadmap view*, *motion command panel*, and *motion sequencing panel*. These components support mainly three user activities for robot programming: exploration and navigation, high-level landmark selection, and OTL motion sequencing. We detail them in the following sections.

1) Exploration and navigation: When the user loads the MFG data and 2D lidar map (Fig. 2 Box 2), the scene view and roadmap view display the robot's environment in an interactive OpenGL scene. Both scene view and roadmap view assist the user with exploration and navigation and provide situation awareness.

From the scene view, the user views the scene as a panorama image that is constructed using the MFG keyframes (Fig. 2 Box 3). For immersive viewing experience, we use a spherical panorama, i.e. a panorama texture mapped on a spherical surface. The user can explore the environment using pan, tilt, and zoom controls in the scene view. The roadmap R is also overlaid in the scene view so that the user can see the navigable paths.

In the roadmap view, the roadmap R is overlaid on the 2D lidar map that is viewed top-down. The roadmap R is constructed from the pose graph P using the algorithm described in Sec. III-D (Fig. 2 Box 4). When the user navigates through the environment, the scene view camera moves along the nodes in R and the spherical panorama is created at the current position of the scene camera. The roadmap view allows translating and zooming for exploration of the environment.

2) High-level landmark selection: When the user explores the environment using the scene view, the user can specify high-level landmarks directly from the panorama image (Fig. 2 Box 7). To specify a high-level landmark, the user uses a mouse to specify a bounding box corner in the panorama image to extract objects of interest. When the user clicks the spherical panorama, an intersection point $\mathbf{X}$ between the sphere and the ray that originates from the mouse position is computed.

Let Π_i^k be the image plane of the camera image I_i^k, and let $\mathbf{C}_i^k$ be the camera center. The intersection point $\mathbf{I}$ between Π_i^k and the line $\mathbf{L} = \mathbf{C}_i^k + (\mathbf{X} - \mathbf{C}_i^k)t$ where $t \in \mathbb{R}$ is computed by $\mathbf{I} = \mathbf{C}_i^k + (\mathbf{X} - \mathbf{C}_i^k)((\mathbf{P} - \mathbf{C}_i^k) \cdot \mathbf{N}_i^k)/((\mathbf{X} - \mathbf{C}_i^k) \cdot \mathbf{N}_i^k)$, where $\mathbf{P}$ is a point on Π_i^k and $\mathbf{N}_i^k$ is its normal. Then, the (x, y) image coordinates of $\mathbf{I}$, i.e. the bounding box corner, in image I_i^k is computed.

When the bounding box is defined, the high-level landmark is highlighted in the spherical panorama. The region inside the bounding box in I_i^k is colored and mapped to the panorama texture $T^k(u,v)$ by using the maps $m_i^x(u,v) = x$, $m_i^y(u,v) = y$, which transforms $I_i^k(x,y)$ to $T^k(u,v)$ by $T^k(u,v) = I_i^k(m_i^x(u,v), m_i^y(u,v))$, where (u,v) is the texture coordinate and is related to the spherical coordinates by $\theta = v\pi$ and $\phi = \pi(2u-1)$, where θ and ϕ are the latitude and longitude angles of the coordinates of the sphere $\mathbf{S} = [r\sin(\theta)\cos(\phi), -r\cos(\theta), -r\sin(\phi)\sin(\theta)]$, where r is the radius.

A label of the landmark can be added through the landmark list and is displayed in the scene view. The high-level landmarks can be saved as a file and can be loaded for later usage.

3) OTL motion sequencing: The user performs motion sequencing using the motion command panel and the motion sequencing panel. The motion command panel contains two list views: the *command list* and *landmark list*. The user selects an item from each list view to form a motion command. Here, we only focus on six commands: *go to*, *start at*, *stop at*, *left at*, *right at*, and *avoid*. New commands can be added per task requirement.

When a high-level landmark is specified by the user, it is displayed as an iconized landmark in the landmark list in the motion command panel. The user selects an item from each list, i.e. a text command and high-level landmark, to form a motion command. When the user presses the *Add motion command* button, an orphan node corresponding to the user-defined motion command is created in the motion sequencing panel.

After the user creates motion commands, which are the building blocks of a motion command sequence, the user can arrange motion commands in the motion sequencing panel to create a motion command sequence M for higher-level tasking. Since high-level landmarks are displayed in the nodes, the user can create high-level commands in an object-oriented fashion. After an orphan node is created from the motion command panel, using a mouse, the user can connect the nodes with directed edges to create a motion sequence. The interface allows the user to add/remove nodes and edges in the graph. When an edge is selected, the user can add transition conditions. In our system, a transition condition is whether a motion command node is visited for a certain number of times (See Fig. 7 for example.) After the user constructs a sequence, the user presses the *Generate path*

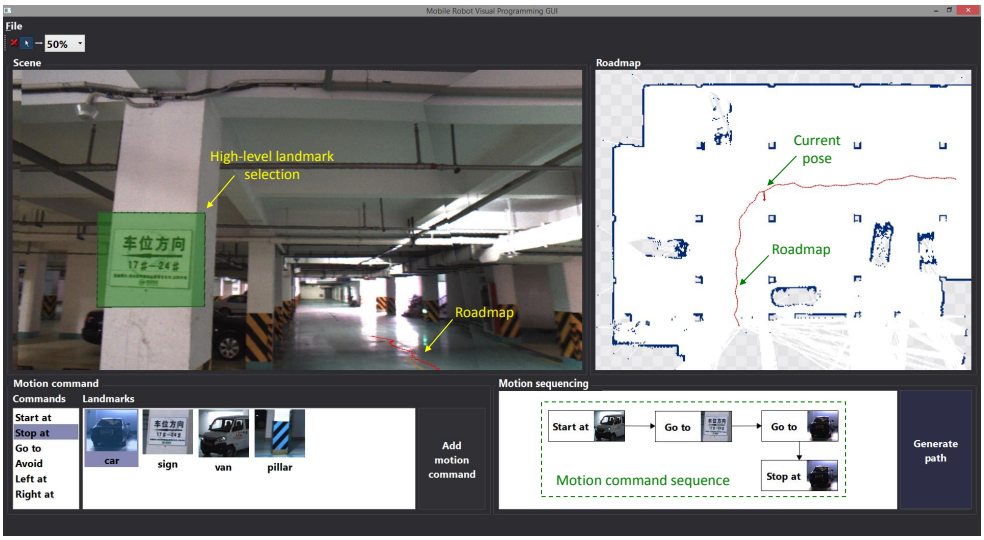

Fig. 3: The GUI of our system. (Best viewed in color.) The GUI mainly consists of four components: the scene view (top-left), roadmap view (top-right), motion command panel (bottom-left), and motion sequencing panel (bottom-right).

Algorithm 1 Pose Graph To Roadmap Preprocessing

Input: Pose graph P, search radius d, number of nearest neighbors k to search
Output: Roadmap R

/* Initialize roadmap */
$R \leftarrow P$ ▷ $O(n_p + m_p)$
/* Build kd-tree */
$T \leftarrow$ KD-TREE-CREATE$(V(R))$ ▷ $O(n_p \log n_p)$
/* Connect near keyposes */
for each vertex $r \in V(R)$ **do** ▷ $O(n_p)$
  /* Find k-nearest neighbors within fixed radius */
  $Q \leftarrow$ KD-TREE-SEARCH(T, r, k, d) ▷ $O(k \log n_p)$
  for each vertex $q \in Q$ **do** ▷ $O(k)$
    if edge $(r,q) \notin E(R)$ **then**
      $R \leftarrow R \cup \{(r,q)\}$
    end if
  end for
end for

button, and the system will compile the motion command sequence M and generate a path S that satisfies the commands (Fig. 2 Box 8). The path S is overlaid on the roadmap view which allows the user to map the high-level commands into physical space at a glance. High-level landmarks associated with the motions are also displayed along the path in the roadmap view.

D. Algorithms

When the user loads the MFG data, the pose graph P is converted into the roadmap R as a preprocessing step using Algorithm 1 (Fig. 2 Box 4). When the user creates the motion sequence and presses the *Generate path* button the path is generated using Algorithm 2 (Fig. 2 Box 8). In the following, for a graph G, we denote the set of vertices and edges by $V(G)$ and $E(G)$, respectively.

In Algorithm 1, roadmap R is first initialized by copying the nodes and edges from P. The value of each node $r_k \in V(R)$ is set as the keypose position $\mathbf{r}_k = -\mathrm{R}_0^{k\mathsf{T}}\mathbf{t}_0^k$, and the value of each edge $(r_a, r_b) \in E(R)$ is set by $distance[(r_a, r_b)] = \|\mathbf{r}_a - \mathbf{r}_b\|$. We build a kd-tree with the roadmap node values $\mathbf{r}_k$. Then, for each $\mathbf{r}_k$, we perform a k-nearest neighbor search within a fixed radius and obtain the set of nodes Q. We add an edge from node r_k to each node in Q if it is not already a neighboring node. This is illustrated in Fig. 4. If we let $n_p = |V(P)|$ and $m_p = |E(P)|$, the time complexity of

Algorithm 2 Path Generation

Input: Motion command sequence M, roadmap R, current node $r_c \in R$
Output: Path S

$m_{curr} \leftarrow head[M]$
while $m_{curr} \neq$ NIL **do**
  /* Handle motion command */
  $c \leftarrow command[m_{curr}]$
  $r_t \leftarrow target[m_{curr}]$
  if $c =$ START **or** $c =$ STOP **or** $c =$ GOTO **then**
    /*graph-search-types */
    $T \leftarrow$ DIJKSTRA(R_c, r_c, r_t) ▷ $O(n_r \log n_r + m_r)$
    $S \leftarrow S \cup T$
    $r_c \leftarrow r_t$
    /* Restore graph edges */
    $R_c \leftarrow R$
  else if $c =$ AVOID **or** $c =$ LEFT **or** $c =$ RIGHT **then**
    /*graph-edit-types */
    $E \leftarrow$ EXTRACT-EDGES(R_c, r_c, c, S) ▷ $O(d_r)$
    for each $e \in E$ **do** ▷ $O(d_r)$
      $distance[e] \leftarrow \infty$
    end for
  end if
  /* Handle transition conditions */
  $m_{next} \leftarrow$ NIL
  $k \leftarrow 0$
  for each edge $e \in outedge[m_{curr}]$ **do** ▷ $O(d_m)$
    if $condition[e] =$ TRUE **then**
      $m_{next} \leftarrow head[e]$
      $k \leftarrow k + 1$
    end if
  end for
  if $k > 1$ **then**
    error "invalid motion command sequence"
  end if
  $m_{curr} \leftarrow m_{next}$
end while

Algorithm 1 is $O(kn_p \log n_p + m_p)$.

Algorithm 2 generates a path S from a motion command sequence M. In a graph point of view, there are two types of motion commands in our system: *graph-search-type commands* and *graph-editing-type commands*. The former includes *start at*, *stop at*, and *go to*, and performs a search in the current roadmap R_c. The latter includes *avoid*, *left at*, and *right at* which modifies edge weights in R_c. For graph-search-type commands, a shortest path search using Dijkstra's algorithm is performed. For the graph-editing-type commands, the edges extracted from EXTRACT-EDGES, which depends on the specified command, are modified. For example, see Fig. 5. Let d_r be the maximum degree of R, and let d_m be the maximum outdegree of M. If we let $n_r = |V(R)|$ and $m_r = |E(R)|$, the graph-search-type commands

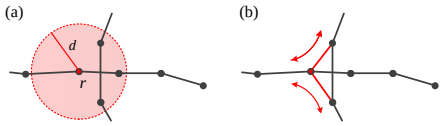

Fig. 4: Pose graph to roadmap. (a) Nearest neighbors search region. (b) Newly added edges and navigable paths.

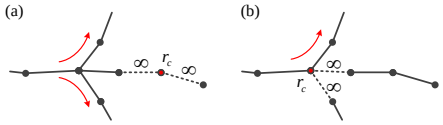

Fig. 5: Modified edge weights and navigable paths using graph-editing-type commands (a) *avoid* and (b) *left at*.

run in $O(n_r \log n_r + m_r)$, whereas the graph-editing-type command runs in $O(d_r)$. Let the number of graph-search-type commands be n_s and let the number of graph-editing-type commands be n_e. Then the worst case time complexity of Algorithm 2 is $O(n_s(n_r \log n_r + m_r + d_m) + n_e d_r)$. If we consider that d_m, d_r, n_s, and n_e are usually small compared to n_r and m_r the time complexity becomes $O(n_s(n_r \log n_r + m_r))$.

IV. Experiments

We have implemented our system in C++. To evaluate the effectiveness of our system, we have tested our system using real world data. We use a set of motion command sequences as input and verify that the generated output paths are correct. We have also tested the runtime of our algorithm and show numerical test results.

A. Data set

Our ground mobile robot that was used to collect data is built on an X80Pro platform. The wheel-based platform has two 12V motors and can move at a maximum speed of 75cm/sec. A SICK TiM571 lidar which has a 25m-depth range, a 270°-angular coverage range, and a scanning frequency of 15Hz is mounted on the platform. On top of the robot, six Point Grey USB3 2.1MP color cameras (BFLY-U3-23S6C-C) are mounted. The cameras are synchronized by an external trigger and capture 960×600 resolution images at a frequency of 1Hz. This frame rate is suitable for our application due to slow robot movement. Here, we present data collected from two environments: office and garage. The size of the environment is shown in Fig. 6, where each chessboard square in the background of the 2D lidar map is $1m^2$. The office data contains 45 keyposes, and the garage data contains 181 keyposes in the pose graph.

B. Roadmap construction and Path generation

We have tested our system using a set of motion command sequences to verify whether our system can generate correct motion paths. For each dataset, we first load the MFG data and 2D lidar map into our system. Then we provide the system with different motion command sequences for testing.

When the MFG data is loaded, our system constructs a roadmap from the pose graph using Algorithm 1. Figs. 6a and 6b show the constructed roadmaps for the office and garage data, respectively. Note that in Fig. 6a, the original

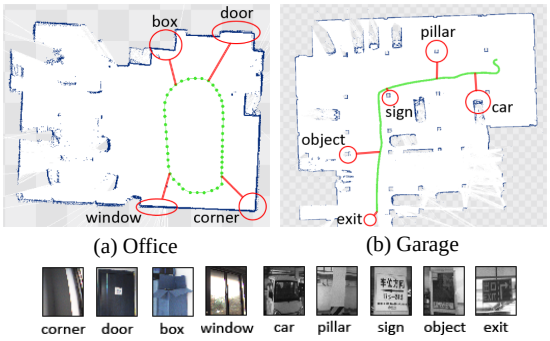

Fig. 6: Roadmaps and high-level landmarks.

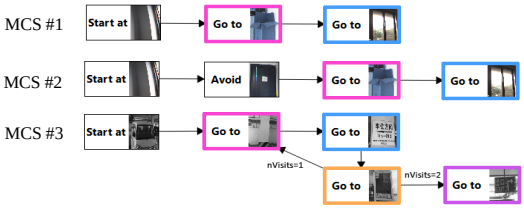

Fig. 7: Sample motion command sequences (MCSs) for testing (Best viewed in color). The color of the motion command nodes correspond to the colors of their resulting path segments in Fig. 8.

sequential pose graph is converted into a loop which allows the user to navigate in different directions.

Next, we use a set of motion command sequences (MCSs) to test Algorithm 2. Fig. 7 shows the motion command sequences that are used for testing: MCS #1 and MCS #2 are used for the office data, and MCS #3 is used for the garage data. Fig. 8a shows the generated path from MCS #1 which contains only graph-search-type commands. The resulting path is the shortest path on the roadmap. We use MCS #2 to test a combination of graph-search-type and graph-editing-type commands. Figs. 8b and 8c show the resulting partial paths where a longer route is taken due to the *avoid* command. Finally, we use MCS #3 to test the transition conditions used in a repetitive motion task. Figs. 8d, 8e, and 8f show the generated partial paths. Due to the transition conditions, the pillar, sign, and object high-level landmarks are visited twice.

C. Runtime test

We test the runtime of our algorithm under different parameter settings. The testing computer is a PC laptop with a 1.9GHz Intel Core i7 CPU and 8GB RAM. The operating system is a 64-bit Windows 8. We test both algorithms in Sec. III-D on the garage data to see how they perform depending on the size of the input pose graph and roadmap. Fig. 9a shows the runtime of Algorithm 1 respect to the pose graph size n_p. For each k nearest neighbor settings, we compute the average of 1000 runs. Fig. 9b shows the runtime of Algorithm 2 respect to the roadmap size n_r. We increase the number of graph-search-type commands n_s from 1 to 5 in a motion command sequence. The results show an average of 100 runs for each n_s. Trends in both Figs. 9a and 9b conform to our complexity analysis.

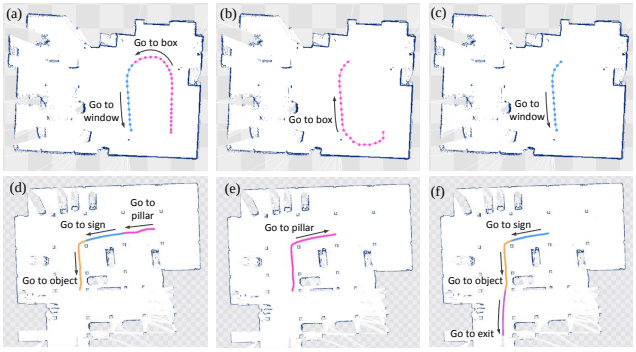

Fig. 8: Path segments from MCSs in Fig. 7 (Best viewed in color).

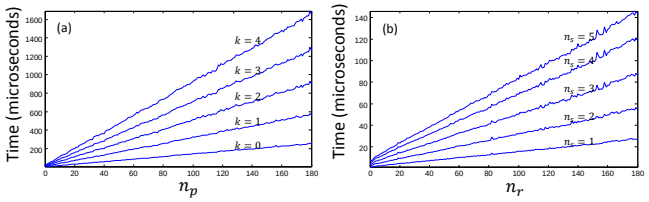

Fig. 9: Runtime results.

V. Conclusions and Future Work

We proposed a new system that allows users to visually program mobile robots. The system used a VR environment constructed from keyframes in vSLAM results to allow users to experience robot working environment and define high-level landmarks associate with OTL motion commands. Our algorithms converted the OTL motion commands to weight point paths to guild robots without overtaxing on human spatial reasoning capability. We presented the data structures, system architecture, user interface, and algorithms. We tested our system using real world data and showed that the generated paths satisfied the motion task requirements. For future work, we would like to extend our method so that the optimal motion sequencing is not restricted to the exact pose graph locations. We will also extend the set of motion commands to support different type of motions.

Acknowledgment

We would like to thank A. Perera and S. Oh for their input during the early development of this project. We would also like to thank C. Chou, H. Cheng, B. Li, S. Yeh, G. Li, M. Treat, R. Liu, and Y. Sun for their input and contributions to the NetBot Lab at Texas A&M University.

References

[1] G. Biggs and B. Macdonald, "A survey of robot programming systems," in *Proc. of the Australasian Conf. on Robotics and Automation*, 2003, p. 27.

[2] Y. Kanayama and C. Wu, "It's time to make mobile robots programmable," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 1, 2000, pp. 329–334.

[3] W. Dai and M. Kampker, "User oriented integration of sensor operations in a offline programming system for welding robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 1563–1567.

[4] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama, "Robot programming by demonstration with interactive action visualizations," in *Proc. of Robotics: Science and Systems (RSS)*, Berkeley, USA, July 2014.

[5] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning to sequence movement primitives from demonstrations," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, September 2014, pp. 4414–4421.

[6] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 2003, pp. 241–248.

[7] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[8] S. M. Lavalle, J. J. Kuffner, and Jr., "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000, pp. 293–308.

[9] J. Denny, R. Sandström, N. Julian, and N. M. Amato, "A region-based strategy for collaborative roadmap construction," in *The 11th Int. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, August 2014, pp. 125–141.

[10] A. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow, "Strategies for human-in-the-loop robotic grasping," in *Proc. of the ACM/IEEE Int. Conf. on Human-Robot Interaction (HRI)*, March 2012, pp. 1–8.

[11] P. Newman, D. Cole, and K. Ho, "Outdoor slam using visual appearance and laser ranging," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2006, pp. 1180–1187.

[12] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," in *Proc. of Robotics: Science and Systems (RSS)*, July 2014.

[13] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research (IJRR)*, vol. 31, no. 5, pp. 647–663, 2012.

[14] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[15] J. Civera, O. G. Grasa, A. J. Davison, and J. Montiel, "1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry," *Journal of Field Robotics*, vol. 27, no. 5, pp. 609–631, 2010.

[16] K. Konolige and M. Agrawal, "Frameslam: From bundle adjustment to real-time visual mapping," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1066–1077, 2008.

[17] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE Int. Symposium on Safety, Security and Rescue Robotics*, November 2011.

[18] Y. Lu and D. Song, "Visual navigation using heterogeneous landmarks and unsupervised geometric constraints," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 736–749, June 2015.

[19] R. C. Arkin, *Behavior-based Robotics*. Cambridge, MA, USA: MIT Press, 1998.

[20] T. W. Fong and C. Thorpe, "Vehicle teleoperation interfaces," *Autonomous Robots*, vol. 11, no. 1, pp. 09–18, July 2001.

[21] D. Lee, O. Martinez-Palafox, and M. Spong, "Bilateral teleoperation of a wheeled mobile robot over delayed communication network," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2006, pp. 3298–3303.

[22] C. Masone, A. Franchi, H. Bulthoff, and P. Giordano, "Interactive planning of persistent trajectories for human-assisted navigation of mobile robots," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct 2012, pp. 2641–2648.

[23] C. Escolano, J. Antelis, and J. Minguez, "A telepresence mobile robot controlled with a noninvasive brain-computer interface," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 3, pp. 793–804, June 2012.

[24] K. Kruckel, F. Nolden, A. Ferrein, and I. Scholl, "Intuitive visual teleoperation for ugvs using free-look augmented reality displays," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2015, pp. 4412–4417.

[25] K. Goldberg, D. Song, and A. Levandowski, "Collaborative teleoperation using networked spatial dynamic voting," *Proceedings of the IEEE*, vol. 91, no. 3, pp. 430–439, Mar 2003.

[26] D. Song and K. Goldberg, "Sharecam part 1: interface, system architecture, and implementation of a collaboratively controlled robotic webcam," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 2, Oct 2003, pp. 1080–1086.