

Received February 20, 2020, accepted February 29, 2020, date of publication March 3, 2020, date of current version March 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2978123

Vivid: Augmenting Vision-Based Indoor Navigation System With Edge Computing

WEI ZHAO¹, LIANGJIE XU², BOZHAO QI³, JIA HU⁴, TENG WANG⁵, AND TROY RUNGE¹

¹College of Agricultural and Life Sciences, University of Wisconsin–Madison, Madison, WI 53705, USA

²Department of Traffic Engineering, School of Transportation, Wuhan University of Technology, Wuhan 430063, China

³Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, WI 53705, USA

⁴College of Transportation Engineering, Tongji University, Shanghai 200092, China

⁵Texas A&M Transportation Institute, Texas A&M University System, San Antonio, TX 78213, USA

Corresponding author: Liangjie Xu (albert_wang01@163.com)


ABSTRACT Indoor localization and navigation have a great potential of application, especially in large indoor spaces where people tend to get lost. The indoor localization problem is the fundamental of an indoor navigation system. Existing research and commercial efforts have leveraged wireless-based approaches to locate users in indoor environments. However, the predominant wireless-based approaches, such as WiFi and Bluetooth, are still not satisfactory, either not supporting commodity devices, or being vulnerable to environmental changes. These issues make them hard to deploy and maintain. In this paper, we present Vivid, a mobile device-friendly indoor localization and navigation system that leverages visual cues as the cornerstone of localization. By leveraging the computation power at the extreme internet edges, Vivid to a large extent overcomes the difficulties brought by resource-intensive image processing tasks. We propose a grid-based algorithm that transforms the feature map into a grid, with which finding the path between two positions can be easily obtained. We also leverage deep learning techniques to assist in automatic map maintenance to adapt to the visual changes and make the system more robust. With edge computing, user privacy is preserved since the visual data is mainly processed locally and detected dynamic objects are removed immediately without saving to databases. The evaluation results show that: i) our system easily outperforms the existing solutions on COTS devices in localization accuracy, yielding decimeter-level error; ii) our choice of the system architecture is scalable and optimal among the available ones; iii) the automatic map maintenance mechanism effectively ameliorates the localization robustness of the system.

INDEX TERMS Last mile delivery, IoTs-based indoor localization and navigation, edge computing for IoTs sensors.

I. INTRODUCTION

Indoor localization and navigation have been an active research area in both academia and industry. Traditional methods, such as maps and instruction signs, are often not convenient enough. For instance, the map of a shopping mall might clearly indicate the location of a shop in the floor plan, but the instruction signs seldom show the specific way to it. Recent commercial and research efforts offer various indoor localization and navigation solutions using different technologies. COIN-GPS [1] addresses the poor signal strength problem of indoor GPS receivers and could achieve an error of less than 10 meters. However, it requires

special hardware support, and cannot be used on commodity devices. WiFi and Bluetooth are the popular choices used to solve the indoor localization problem. Recent research has shown that Bluetooth-based solutions can achieve a median error of 1.5m [2], while the WiFi-based commodity device-compatible approaches are reported to have achieved 2m [3]. Although empirically such accuracy is sufficient for indoor navigation, it has been pointed out that the received signal strength (RSS) method they use is extremely vulnerable to the environment with interferences [4], [5]. In addition, these approaches are not easy to deploy and manage because of the large amount and complexity of the devices. They do not provide the orientation information of the device, either. Thus, other technologies such as the inertial motion sensors will have to be used as a complement to realize navigation.

The associate editor coordinating the review of this manuscript and approving it for publication was Heng Wang .

Over the years, mono-camera tracking techniques can run on a normal personal computer without GPU acceleration in real-time and with a reasonable accuracy. For example, ORB-SLAM claims itself to have achieved an error as little as 1% of the map dimension [6]. In addition, the output of the SLAM (simultaneous localization and mapping) systems already contains the orientation of the camera, which eliminates the complexity of sensor fusion. Vision-based SLAM seems to be a rather promising technique to apply to indoor localization. However, there are a few challenges needed to be overcome before we could adapt it to an indoor navigation system. On one hand, if we run SLAM on the smart devices, their CPUs are very likely not as powerful as those on personal computers, hence can hardly provide satisfactory responsiveness. The power consumption would be an issue as well. In addition, a map itself that can easily exceed a few hundred megabytes in size could be hard for some smartphones to store. It also takes a long time to download, undermining the battery life. On the other hand, if we consider running SLAM on a centralized server, the continuous images captured by the users' cameras need be streamed to the server. It is suspicious whether the computing power is scalable enough to accommodate a large number of concurrent users in a large public indoor space, let alone the limited network bandwidth. Another challenge of vision-based SLAM systems is how to handle the dynamic environment changes. The map may be constructed with no crowd in the space, but people can be all over the place in deployment. The environment texture can also change due to reasons like a shop uses a new decoration. Besides, the lighting condition could also change in a day.

A First Look at Vivid: To solve the above-mentioned problems, we propose Vivid, a vision-based system that leverages the edge computing paradigm to provide accurate, mobile-friendly and privacy-conserved navigation services in indoor environments. Our system utilizes routers that possess relatively strong computing power as the edge nodes that provide both the WiFi access points (APs) for the users and the computing resource for the vision-based localization algorithms. We find this perfectly solves the dilemma that the user devices are too weak to run image processing tasks while centralized servers are likely to have difficulty scaling to support a large number of concurrent users. We use a cloud server to coordinate the edge nodes. The cloud-edge architecture brings benefits to the system from two perspectives. First, image sequences are processed on edges to remove any sensitive information before uploading to the cloud. Second, the cloud server can manage and coordinate different edge nodes to support inter-node collaboration and facilitate navigation across multiple edge nodes. With deep learning techniques, the system is able to overcome the shortcomings of SLAM and better adapt to environmental changes. To be specific, the contributions of this paper include:

- i) an accurate and scalable vision-based indoor navigating system consisting of a client app, edge node services, and a cloud coordinator server;

- ii) the grid-map based algorithm that transforms the SLAM map to a floor plan and projects the SLAM coordinates and orientations onto the floor plan, which facilitates route planning;
- iii) a deep neural network (DNN)-based mechanism that enables the system to automatically adapt to environment lighting and textural changes without the interference of dynamic objects;
- iv) security of user privacy due to the information collection and processing offloaded to the network edge and our deliberate design.

The rest of this paper is organized as follows. section II describes the design of the system. section III highlights the algorithms and techniques we propose and leverage in the system. section IV demonstrates the evaluation results and analysis. section V discusses the limitation and outlook of our system. section VI introduces the background and lists some related work, and section VII concludes the paper.

II. SYSTEM DESIGN

A. MOTIVATION

As is mentioned in the previous sections, Vivid is a vision-based indoor navigation system. In particular, we use SLAM as our cornerstone for localization, which, on the one hand, needs streams of images as the input, and on the other hand, consumes a large amount of CPU resources. Given these conditions, we deem the traditional client-cloud server paradigm not suitable for our purpose, since the bandwidth between the local area network (LAN) and the cloud server could probably be the bottleneck for the number of clients to scale up. In fact, as we will see in section IV, even with a centralized server in the local network, the quality of service (QoS) of the network could still be insufficient when we increase the number of concurrent users. Directly running SLAM on mobile devices is also out of the question. Although mobile devices can run SLAM, this approach is either barely scalable or unfriendly to the battery life. Moreover, the users would like to perform localization with the existing map, which means that the map has to be downloaded to the device beforehand, which can take up large storage spaces and can be time-consuming. We will report the sizes of the map files in section IV.

Edge computing, in this sense, is the most compatible and adaptive choice that we have. Inspired by ParaDrop [7], [8], we assume the Wi-Fi routers to possess computing power equivalent to a personal computer (PC). These routers will act as the edge nodes that directly handles the data streams from the devices connected to it. However, each edge node has its own range of Wi-Fi coverage, multiple edge nodes are applied to cover large indoor spaces. By supporting this purpose, we produce another problem, that is to support navigation across multiple edge nodes. Therefore, we use a cloud server to manage and coordinate the edge nodes so as to support inter-node collaboration and provide maintenance service. At this point, the architectural design of Vivid has been formed.

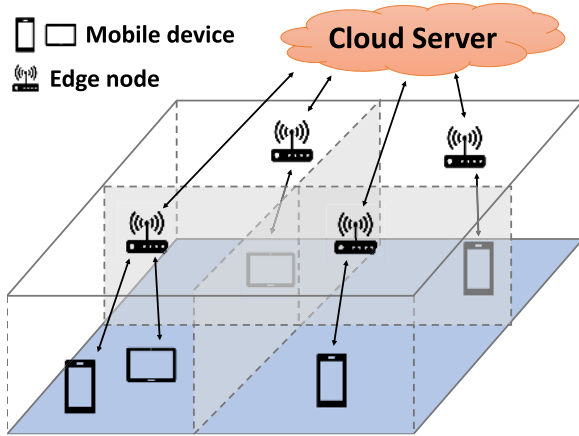


FIGURE 1. The architecture of Vivid.

B. OVERVIEW

Fig. 1 shows the high-level architecture of Vivid. From bottom-up, the system consists of three parts: i) mobile devices; ii) edge nodes; and iii) a cloud server. In this figure, the space is covered by four edge nodes, each communicates with the mobile devices connected to it. Note that the actual space coverage of a specific edge node can be affected by various factors, such as walls, the interference in the environment, and the relative signal strength to its neighbors. Above the edge nodes is a cloud server that oversees the edge nodes.

1) MOBILE DEVICES

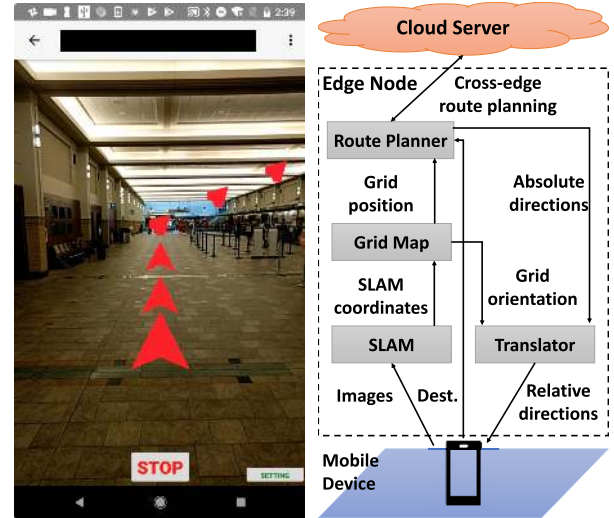
The mobile devices run an application that has three major functionalities: i) connecting to the desired destination and transmitting the image sequence captured from the camera to the edge node it is connected to; ii) receiving the responses from the edge node that contain the directions of next step; and iii) displaying the directions by an navigation UI on the screen for the user. This UI at navigation time is shown in Fig. 2a. No complex computation task is finished by the application.

2) EDGE NODES

Fig. 2b shows the major modules of the edge node service application, as well as the procedure of handling a navigation request. With the edge node initialized (section III-A), the mobile device sends the destination to Route Planner and starts streaming image sequence to the edge node. For each image, it sent to SLAM for the SLAM coordinates of the camera, and projected onto the grid map (section III-C). The grid position is used to plan the route. If the destination is in the range of another edge node, the coordination of the cloud server is needed (section III-D). Finally, the sequence of direction given by Route Planner is translated back to the directions relative to the camera's orientation.

3) CLOUD SERVER

The cloud server has two major responsibilities: global route planning and map maintenance, which will be introduced in detail in section III-D and section III-E respectively. The



(a) The UI of the mobile device app (b) The structure of the edge node service application

FIGURE 2. An overview of Vivid's indoor navigation.

cloud server should have a GPU since we will perform object detection on it. The global route planning is performed synchronously on the edge node request, while the map maintenance mechanism that needs the GPU is carried out in an asynchronous manner.

III. STARTING Vivid

A. INITIALIZATION

1) SLAM MAP

Different from the typical use case of SLAM, where localization and mapping happen at the same time, we use pre-constructed maps to relocalize the camera. The map can be constructed by the administrator beforehand using more accurate devices such as the stereo camera, and uploaded to the edge node. Since we assume that the device's switch of the edge node is done by the low-level mechanism and opaque to the user, the range of the map stored in an edge node should be larger than the area it is planned to serve, in case that the user is out of the map range of the node while have not switched to the next one.

2) GRIDDING INFORMATION

After a SLAM map is constructed, information that helps align the grid map with the SLAM map, which includes the x and y vectors of the grid system under the SLAM coordinates. These vectors represent the column and row directions and their cross product should point upwards. They can be trivially obtained by subtracting the SLAM output of two camera positions. The number of rows and columns of the grid, m and n , should also be determined.

3) DESTINATIONS

While constructing the map or utilizing relocalization afterwards, the administrator may record some SLAM coordinates

T with a unique string key s in a pair $\langle s, T \rangle$ that describes the destinations and their places in the SLAM. After the grid map is established (section III-B), they will be transformed to $\langle s, \mathbf{u} \rangle$, where $\mathbf{u} = (x_u, y_u)$ is the corresponding grid coordinate of T (section III-C). Such pairs will be stored in a hash table for future lookup.

B. MN-SCALED GRIDGING

The maps of SLAM systems are usually designed for localization and motion tracking only, which means that they tend to lack the primitives for navigation. A SLAM map usually contains only the keyframes and points of interest like features and points with large intensity gradient [6], [9] in the three-dimensional Euclidean space. If the map is constructed with a monocular camera, it hardly reflects the real-world scale. However, real-life navigation needs to store domain knowledge about the terrain, such as viable ways, barriers and destinations so as to plan the route. In our particular case, the navigation maps on neighbor edge nodes should also support smooth cross-edge (global) route planning.

With these requirements in mind, we design a grid map. It is essentially an $m \times n$ boolean matrix that spans the horizontal plane within the indoor space. We let *true* cells represent a place where people can appear, and *false* cells stand for places at which people cannot arrive. We propose the mn-Scaled Gridding algorithm that leverages the keyframes in the SLAM map to fill the matrix. We first make three assumptions: i) we handle one flat floor at a time only; ii) the keyframes are approximately at the same height to the floor; and iii) at least one keyframe is saved for each place where people can be present. With these assumptions, we perform surface fitting on the keyframe coordinates, project them onto the grid, and set the corresponding matrix item to *true* if there is at least one keyframe projected to the cell. The procedure is detailed in Algorithm 1 and demonstrated in Fig. 3 with monocular ORB-SLAM2 [6]. Note that in this algorithm, *surfaceFitting* gives a normal vector of the fitted surface and *normalize* normalizes a vector. These are considered as trivial subprocedures thus not elaborated.

To ensure the globality of the grid map, we take special care to avoid an anomalous frame, such as a sudden occlusion of the camera lens or a series of blurred frames, from causing keyframe repeatedly inserted at a single point on the map. We solved this problem by keeping the indices across an interval of frames, which can be regarded as a sliding window moving across all the keyframes. Our algorithm will decide if the current frame is spawned as a new keyframe at a new/nearby grid on the map. We developed a keyframe selection mechanism to remove redundant keyframes in the process of building the grid map. We developed the following rules for adding a new keyframe to the grid map.

- i) At least 10 frames must have passed from the last global re-localization.
- ii) The cell is false, or more than 10 frames have passed from last keyframe insertion to a cell on the grid map.

Algorithm 1 mn-Scaled Gridding

Input:

- n_{kf} : Number of keyframes
- $P[1..n_{kf}, 1..3]$: Cartesian coordinates of the keyframes in the SLAM system
- \mathbf{x} : Vector aligned with the column direction of the grid map in SLAM coordinates
- \mathbf{y} : Vector aligned with the row direction of the grid map in SLAM coordinates
- m : Maximum length of space in the column direction in a unit
- n : Maximum length of space in the row direction the same unit

Output:

- $G[1..m, 1..n]$: Boolean matrix representing the grid map

```

1: Initialize  $G[1..m, 1..n] \leftarrow \{false\}$ 
2:  $\mathbf{n} \leftarrow \text{surfaceFitting}(n_{kf}, P)$ 
3: if  $\mathbf{n}_s \cdot (\mathbf{x} \times \mathbf{y}) < 0$  then
4:    $\mathbf{n}_s \leftarrow -\mathbf{n}_s$ 
5: end if
6:  $R_{sg} \leftarrow [\text{normalize}(\mathbf{x}) \text{ normalize}(\mathbf{y}) \text{ normalize}(\mathbf{n}_s)]$ 
7: for  $p$  in  $P$  do
8:    $p \leftarrow R_{sg}p$ 
9: end for
10:  $x_{max}, y_{max} \leftarrow \max(P[:, 0]), \max(P[:, 1])$ 
11:  $x_{min}, y_{min} \leftarrow \min(P[:, 0]), \min(P[:, 1])$ 
12: for  $p$  in  $P$  do
13:    $x_g, y_g = m \lfloor \frac{p[0] - x_{min}}{x_{max} - x_{min}} \rfloor, n \lfloor \frac{p[1] - y_{min}}{y_{max} - y_{min}} \rfloor$ 
14:    $G[x_g, y_g] \leftarrow true$ 
15: end for

```

- iii) Current frame tracks at least 50 points in this cell.
- iv) Current frame tracks less than 90% points than last inserted frame in that cell.

SLAM has its own redundant keyframe removal algorithms, besides that, we developed the abovementioned rules to further filter out frames that have similar features. Based on the outputs, we provide additional information in the app to alert the user when they are building the map. The user can view the selected keyframes on the grid map (similar to Fig. 3), and when there are many frames selected in a single cell, the app will notify the user. Besides, the user can also know which cell needs more frames on the map.

After applying this algorithm, we obtain a grid map that eliminates the complicated data structures for domain concepts, but clearly indicates where people can pass and where they cannot, which makes it simpler to plan the route.

C. GRID PROJECTION

With the grid map established, the system should also be able to convert the SLAM coordinates to our grid coordinates for the route planning purpose. We call this process the grid projection. As is shown in Fig. 4, we project both the position

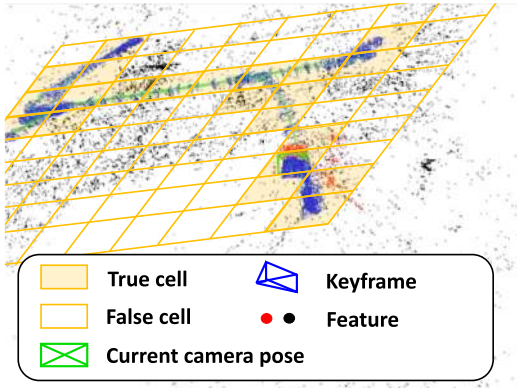


FIGURE 3. A demonstration of mn-scaled gridding.

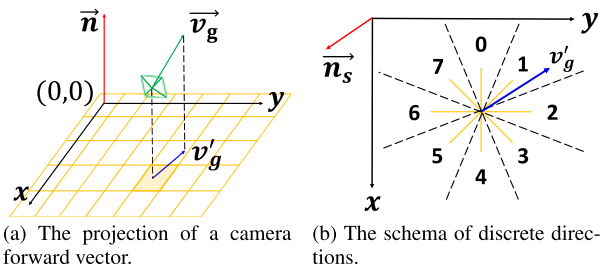


FIGURE 4. Orientational projection.

and the orientation to the grid. The positional projection is carried out in the same way as we project the keyframes onto the grid plane, described in Line 7-13 of Algorithm 1, where (x_g, y_g) is the corresponding cell of the given SLAM coordinates p .

To better facilitate navigation, the system also has to convert the SLAM orientation into the grid map system. Since we assume a planar map, we can project the three-dimensional orientation onto our grid plane. In addition, given that each cell in our grid map has only 8 neighbors, we can further simplify the orientation into 8 directions numbered from 0 to 7. As Fig. 4a shows, given the forward vector of the camera in SLAM coordinates v , the corresponding vector in the grid map system is $v_g = R_{sg}v$, where R_{sg} is obtained in Algorithm 1. v_g is then projected onto the grid plane, which gives v'_g . The output direction d is determined according to the direction of this projection and the schema, which is shown in Fig. 4b.

D. ROUTE PLANNING

As we have established the grid map, and have been able to relate the SLAM output with the coordinates in the grid map, we can leverage it to plan the navigation route. The goal of route planning is to enable local (within the edge node) and global (cross-edge) search for the shortest path based on the relatively independent grid map on each edge node. To achieve better performance, we mainly leverage edge computing platforms and only upload processed information to the cloud when necessary. Compared to cloud

Algorithm 2 Local Graph Modelling

Input:

$M[1..m, 1..n]$: Grid map

h, w : Actual height and width of the grid cell

Output:

$G(V, E)$: Graph representing the graph of the *true* cells and their connections

```

1: Initialize sets  $V, E \leftarrow \{\}$ 
2: for  $i, j \in \{1..m\} \times \{1..n\}$  do
3:   if  $M[i, j]$  then
4:      $V.insert(v \leftarrow (i, j))$ 
5:     for each true neighbor  $(x_u, y_u) \leftarrow u$  of  $v$  do
6:       if  $|x_u - i| + |y_u - j| = 2$  then
7:          $e_{u,v} \leftarrow \sqrt{h^2 + w^2}$ 
8:       else
9:          $e_{u,v} \leftarrow (x_u \neq x) ? h : w$ 
10:      end if
11:       $E.insert(e_{u,v})$ 
12:    end for
13:  end if
14: end for
15: Construct graph  $G(V, E)$ 

```

computing, edge computing provides lower latency, greater responsiveness, and more efficient use of network bandwidth. With edge computing, data can be analyzed at the local device level, closer to where it is being generated, and only upload data to the cloud when necessary. With this setup, less data is transmitted from local devices via a network to a cloud, thereby reducing network traffic bottlenecks. Moreover, lower latency can be achieved from two aspects. First, less data is transmitted to the cloud, so the transmission time is reduced. Second, data can be analyzed locally, we don't need to wait for the results generated from the cloud. Back to our application scenarios, continuous frames are pre-processed locally and so not all the raw frame data need to be uploaded to the cloud. It helps reduce significant amount of bandwidth and allows our system to support more users. Based on our measurements, with the support of edge compute, the latency for single user case reduced 50 times.

We discuss the local and global scenarios separately.

1) LOCAL ROUTE PLANNING

The grid map that we obtain in section III-B is a $m \times n$ boolean matrix with each *true* cell representing a viable position. We also know the real-world scale of a cell. We design Algorithm 2 to transform the problem model into a typical undirected weighted graph. With this graph, it is trivial to utilize the well-known Dijkstra or Bellman-Ford algorithm to solve for the shortest path given origin s and destination t .

2) GLOBAL ROUTE PLANNING

We use a similar approach as local route planning, modeling the problem as a shortest path search problem in a graph. In order for the cloud server to coordinate the path planning across all the edge nodes, we make a few reasonable assumptions on the information that the cloud server possesses, which would significantly simplify the problem. After that, we will illustrate the solution.

Assumption 1: The cloud server has the grid map of every edge node, which we denote as E_i such that $i \in \{1..N\}$ where N is the total number of edge nodes. The cell height and width h_i and w_i are always provided. A hash table for destinations H_i is attached with each grid map E_i that stores pairs $\langle s, u \rangle$ where s is the name of the destination, and $u = (x_u, y_u)$ is the coordinates within that grid map.

Assumption 2: If there exists a way to enter directly from one grid map to another, then at the edge of both grid maps exists some special true cells that represent the entrance to the other map. We call such a special connection cell, demonstrated in Fig. 5a, denoted by $c_k^{(i)}$, meaning the cell is the k -th entrance of grid map i to another map. These cells can be marked after map construction together with the destinations. We assume that the cloud server has mapping $\Phi_c[c_\alpha^{(i)}, c_\beta^{(j)}] \rightarrow \{true, false\}$ that determines whether two connection cells on node i and j respectively are adjacent.

Assumption 3: With each grid map, the cloud server can calculate and store beforehand the shortest distance between each pair of connection cells on that map. The construction of the graph can be achieved by Algorithm 2. The difference is that the problem that we are dealing with here is a multi-source shortest path problem. Therefore, the Floyd-Warshall algorithm can be applied. Therefore, we assume that for each grid map i , the cloud server already stores $W_i[c_\alpha, c_\beta]$ as the shortest length from c_α to c_β .

Solution: With all the assumptions above, we are able to model the problem as a graph. We regard each connection cell as a vertex in the graph. If two cells have the same grid attribution, they have an edge with the shortest path length being the weight. If two vertices are adjacent cells in neighboring grids, there is also an edge between them. For convenience, we set the edge weight between two adjacent cells in neighboring grids to be the average of the height and width of both cells. The details of the modelling are described in Algorithm 3. When global path planning is requested, the origin and the destination will both be added to the graph as vertices. Their shortest path lengths to the connection cells of the grid they belong to will be the edge weight. Fig. 5b shows an example of the graph model generated by our algorithm. With this graph, it is trivial to run the Dijkstra or Bellman-Ford algorithms to obtain the shortest path from the origin to the destination. The cloud server sends the connection cell. From the grid the user should enter a neighbor grid map, the cloud server sends the connection cell to the edge node that serves the user. And finally, the local shortest route to that connection cell will be shown for the user.

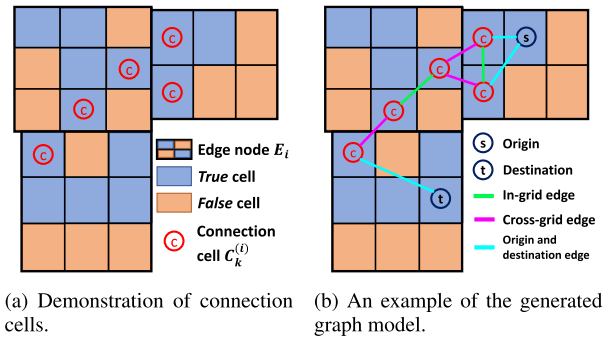


FIGURE 5. Remote route planning.

3) DIRECTION TRANSLATION

The output of route planning is a series of discrete directions in according to the schema of the grid map (section III-C). They have to be translated to the mobile device's relative directions. We define the forward direction of the camera as 0, and 1-7 are numbered clockwise similar to Fig. 4b. Let $D_g[1..n_d]$ be the planned directions and d be the current direction, then the direction sequence relative to the camera $D_c[1..n_d]$ is given by:

$$D_c[i] = D_g[i] - d \mod 8, \quad \forall i \in \{1..n_d\} \quad (1)$$

The mobile device application presents these relative directions to the user.

E. AUTOMATIC MAP MAINTENANCE

As is mentioned in section I, one of the drawbacks of applying SLAM is that it could be hard to promptly update the map according to the visual changes of the environment. An example would be the change in the shop window of a clothes shop in a mall, which makes the corresponding part the features or intensities in the SLAM map useless. One may argue that since the nature of SLAM is to localize and construct the map at the same time, it is possible to use the image sequences uploaded by the mobile devices to fix the map change. The problems with this approach are: first, the features of dynamically occurring objects like people can also be recorded; second, the performance of edge nodes can be affected by the extra calculation introduced by mapping. Therefore, we need to distinguish whether a user uploads frames/video contain dynamic objects – moving objects, and move this task from edge nodes to the more powerful cloud server. Since the map maintenance is on the cloud, we can safely leverage deep learning-based object detection performance without worrying about the local practical working status. We will use the object detector as a black box that takes in an image and spits whether the image contains objects belonging to the given set of classes.

Eventually, we design an automatic map maintenance mechanism as demonstrated in Fig. 6. As the mobile device streams images to the edge node, if the SLAM localization is successful, the edge node buffers the most recent images

Algorithm 3 Remote Graph Modelling**Input:**

$C_i[1..n_c^{(i)}]$, $\forall i \in \{1..N\}$: Connection cells of each grid map
 $h_i[1..N]$, $\forall i \in \{1..N\}$: Actual cell height of the grid map
 $w_i[1..N]$, $\forall i \in \{1..N\}$: Actual cell width of the grid map
 $\Phi_c[c_\alpha^{(i)}, c_\beta^{(j)}]$: Whether $c_\alpha^{(i)}$ and $c_\beta^{(j)}$ of grid maps i and j are adjacent to each other
 $W_i[c_\alpha, c_\beta]$: Shortest path length between c_α and c_β in grid map i

Output:

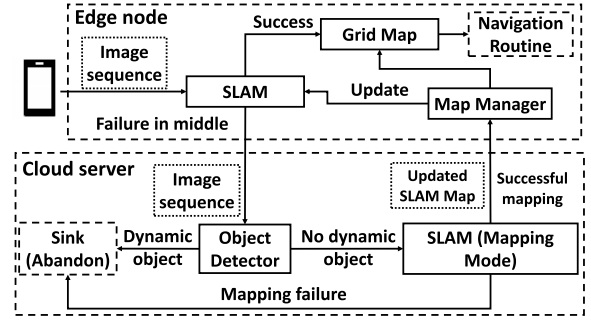
$G(V, E)$: Graph representing the graph of the connection cells and the weights between them

```

1: Initialize sets  $V, E \leftarrow \{\}$ 
2: Initialize  $W[e]$  such that  $e \in E$ 
3: for  $i \in \{1..N\}$  do
4:   for  $j \in \{1..n_c^{(i)}\}$  do
5:      $V.insert(C_i[j])$ 
6:   end for
7:   for distinct  $j, k \in \{1..n_c^{(i)}\}^2$  do
8:      $e(C_i[j], C_i[k]) \leftarrow W_i[C_i[j], C_i[k]]$ 
9:      $E.insert(e(C_i[j], C_i[k]))$ 
10:  end for
11: end for
12: for distinct  $i, j \in \{1..N\}^2$  do
13:   for  $\alpha, \beta \in \{1..n_c^{(i)}\} \times \{1..n_c^{(j)}\}$  do
14:    if  $\Phi_c[C_i[\alpha], C_j[\beta]]$  then
15:       $e(C_i[\alpha], C_j[\beta]) \leftarrow \frac{h[i]+h[j]+w[i]+w[j]}{4}$ 
16:       $E.insert(e(C_i[\alpha], C_j[\beta]))$ 
17:    end if
18:  end for
19: end for
20: Construct graph  $G(V, E)$ 

```

of some time length. When a localization failure occurs, the edge node keeps collecting images for some time, and send these images along with the buffered ones to the cloud server. The cloud server first uses the object detector to discover the dynamic objects in the image frames. If people are found in the image, the part of image within the bounding box is immediately deleted at first for privacy. Although the images with other dynamic objects can be leveraged to perform future analysis, we abandon them as well, since we are interested in the environment texture. The image sequence without dynamic objects is sent to SLAM with mapping mode activated. The cloud server has copies of all the SLAM maps on the edge devices and can directly load them to SLAM for map maintenance. The SLAM system handles the images with the normal routine. If the map is successfully updated, it will be pushed to edge nodes. The *Map Manager* module is responsible to load this map to SLAM and update the grid map with mn-Scaled Gridding (section III-B).

**FIGURE 6.** Automatic map maintenance.

The map maintenance job is consisting of two parts. First, when there are some permanent changes of the surrounding environments, like new decorations, remodel of the indoor environments, the SLAM map changes accordingly with new image data stream, and the grid map will be updated with the new keyframes generated from SLAM map. We could use some deep learning techniques to distinguish the changes automatically, and we left this as future work. Second, when there are dynamic objects in the video stream, during the process of building the map, we applied deep learning algorithms to detect, locate, and remove the object from the frame. Doing this can help the system get rid of unwanted noisy pixels.

Automatic maintenance cycles and training samples are defined with a dynamic window according to the usage of the computing power and bandwidth. The real-time navigation for users is always the top priority. So, we set up the trigger when the usage rate is lower than 30%. And this maintenance tasks mostly happen at night.

IV. EVALUATION

To evaluate, we implement our system with an off-the-shelf pull request of ORB-SLAM2 [10] that supports map save and load [11] by leveraging the Boost Serialization library [12]. This particular SLAM implementation is chosen only because of the engineering convenience. Other monocular SLAM systems should also fit with proper engineering of adaptation. The specific settings for each part of the evaluation will be detailed in the corresponding section.

A. ACCURACY

Although most SLAM works have their own accuracy claim, the localization accuracy of our system still needs evaluation, since we apply the gridding algorithm, whose impact on the accuracy is non-intuitive. In this experiment, we measure the localization accuracy in scenarios having different feature diversities with various grid density. We will discuss the results in Sec. V.

Setup: Fig. 7a and 7b demonstrate the indoor environment and the corresponding floor plan of this experiment. We first establish a SLAM map of the space, then apply the mn-Scaled Gridding algorithm to generate the grid map. In order for the edges of the space and the grid to be consistent, the camera was taken to the space edges where SLAM keyframes are

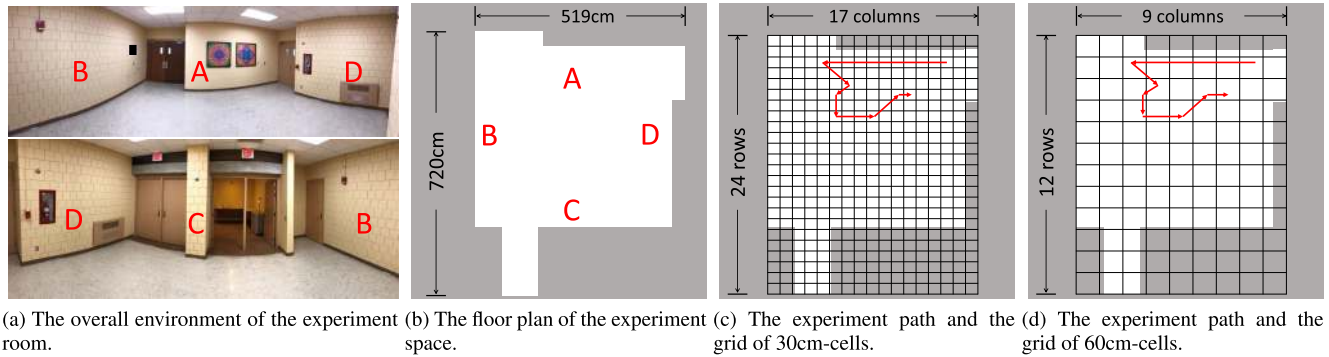


FIGURE 7. The configuration of the accuracy experiment.

recorded. The width and height directions of the grid map are also strictly calibrated with those of the space. These guarantee that the grid fits the floor plan in size and orientation, thus the experiment results are precise to our best effort.

The SLAM implementation that we adopt, ORB-SLAM2 in monocular mode, is claimed to typically have an indoor accuracy below 1cm [6]. However, since the density of keyframes is not mentioned, and that the validity of each position in the grid depends on whether there is at least one keyframe in it, the grid cell should not be too small, or a position can be invalid even if the camera passed through it during map construction due to the lack of keyframes inside. We therefore carry out preliminary experiments, from which we empirically decided that 30cm of cell width and height is absolutely safe for this particular case. Eventually, as Fig. 7c and 7d have shown, we choose approximately 30cm and 60cm for the width and height of a grid cell. Note that in our experiment setup, 30cm and 60cm cells are selected in a similar way. First, existing SLAM systems could achieve a similar or even smaller map resolution, which means it is reasonable for us to choose a cell size as small as 30cm [6], [43]. Besides, we also considered several other cell sizes, e.g. 15cm, 45cm, 90cm, as comparison groups to evaluate system performance. Since the common human walk step/stride is ranging from 30cm to 60cm [41], [42], these two setups should be sufficient to include most common scenarios in real world. According to our loss function algorithm, a larger grid size has a higher possibility to achieve a better localization performance. In Table 1, the RMSE of 60cm cell is 0cm, which means when the cell is equal to or greater than 60cm, our system can achieve a 100% localization accuracy (please refer to later sections for evaluation results). Hence, the system can achieve a better performance with a 60cm cell size comparing to 30cm. Moreover, our algorithm is able to provide good localization accuracy with grid width between 30cm and 60cm, which makes this algorithm practical in real world application scenarios.

Due to the aspect ratio of the floor plan, the cells are not perfectly square. We will consider the actual length in the calculation.

For both settings, we select the same path, which is marked in both figures. This path is picked because there is a corresponding reference on the floor for the ground truth. We use

TABLE 1. Results of the accuracy experiment.

Cell length (approximate) (cm)	30	60
Localization success frequency (%)	87.5	86.4
Accurate localization frequency (%)	78.6	100
RMSE (cm)	17.21	0
Maximal error (cm)	42.8	0
Orientation accuracy (%)	100	100

a Google Pixel 2 smartphone as the guest device and a laptop PC as the server, which are connected via the Wi-Fi hotspot of the laptop. The Pixel 2 and the camera used in map construction are calibrated using the OpenCV camera calibration model [13] with the same parameters. During the experiment, the smartphone is moved along the path facing the direction of the next position. At each grid cell, the grid coordinates computed by the server and the corresponding ground truth are recorded. The experiment is conducted three times with each configuration.

Result: Table 1 shows the result of both settings. The relative frequency of successful localization is the percentage of the records with a successful localization from the server, while the relative frequency of accurate localization is the number of localization records to the number of successful localization. We define an accurate localization as the system output of grid coordinates being exactly the same as the ground truth. The root-mean-square error (RMSE) measures the derivation of localization in terms of real-world distance from the ground truth. It is calculated by:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^m ((h(x_i - x_i^{(g)}))^2 + (w(y_i - y_i^{(g)}))^2)}{m}} \quad (2)$$

In this equation, h and w are the actual height and width respectively of the grid cell, (x_i, y_i) is the server output of the row and column coordinates, $(x_i^{(g)}, y_i^{(g)})$ is the corresponding ground truth, and m is the number of records of successful localization. The maximal error is the maximal Euclidean distance between the centers of the localization result and the ground truth.

From the table, we find that the 30cm group yields 87.5% successful localization, among which 78.6% is fully accurate, while this number for the 60cm group is 100%. The RMSE of the 30cm grid is 17.21cm, while it is 0cm with the 60cm grid.

One special discovery is that, the maximal error is 42.8cm in the 30cm group, which is occurred when the smartphone was localized at the diagonal neighbor of the ground truth. We also compare the orientation in the grid plane calculated by the system with the ground truth, and find them completely the same.

Analysis: We conducted this experiment with two purposes: to demonstrate the system's accuracy, and to understand the parameter sensitivity of the n-Scaled Gridding algorithm.

From the result, we find that the localization success frequency for both groups are close. In fact, whether or not a localization is successful depends solely on SLAM. By design of the gridding algorithm, as long as the SLAM subsystem can output the SLAM map coordinates, there is always a corresponding grid position. Therefore, the unsuccessful positioning is most likely due to SLAM's failed localization, which will be discussed later. In terms of successful positioning, however, shows that with 60cm grid, the output of the system perfectly matches the ground truth. Even when the grid is condensed with 30cm cells, the furthest error lies in the diagonal neighbor of the ground truth. The error is probably resulted from the error of both the SLAM localization itself and the inaccurate alignment of the grid map and the SLAM map. In terms of the direction, we find that under the 8-direction schema, the results are completely accurate for both groups. Therefore, despite the error, this experiment has verified that the system can achieve a positional accuracy of at least 60cm and a perfect orientational accuracy with 8 directions.

B. SCALABILITY

We conduct an experiment to compare the scalability of the proposed edge computing architecture and a typical configuration with a centralized server by gradually increasing the number of users simultaneous requesting service and measure the intervals between successive responses observed by each user with the centralized server, one and two edge nodes respectively.

Setup: We have two Intel® NUCs each with an Intel® Core™ i7-6770HQ CPU, an IEEE 802.11ac 2 × 2 wireless network interface card (NIC), and a 10/100/1000Mbps ethernet NIC, installed with Ubuntu 18.04 and our edge node application. The mobile application is installed on a total of 10 smartphones, consisting of 4 Google Nexus 5, 2 Google Pixel and 4 Google Pixel 2, all of which running Android 8.0. The mobile application is configured to stream the images captured from the main camera at 6 frames per second (fps) and 640 × 480 resolution to the edge node or server the device is connected to. Two PCs each with an IEEE 802.11ac wireless NIC are leveraged to simulate the image streams generated by different numbers of smartphones when necessary. Only the smartphones collect data. In the double edge node configuration, both NUCs act as the edge nodes, each connecting to half of the devices directly via its own Wi-Fi hotspot, while in the single edge node setting, only one NUC

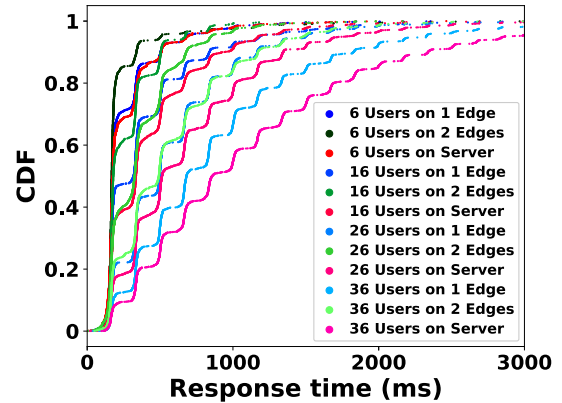


FIGURE 8. The CDF of the time intervals between responses.

covers the entire space. In the centralized server configuration, the NUC is the centralized server. The smartphones and PCs are connected to a 802.11ac router, which then connects to the centralized server via a switch. Each setting of the experiment is conducted in the same 8m × 5m rectangular room. The single node and the router is placed at the center, while the two edge nodes divide the space into 4m × 5m halves and each is placed at the center of its half. Each smartphone represents a user and records the timestamp of each request sent to the server and the timestamp of the corresponding response. The NUC records the CPU usage of the server process from the *top* command.

Result: Fig. 8 shows the cumulative frequency of the time intervals between successive responses from the server and the edge node(s) with different number of concurrent users. We also refer to such time intervals as the user waiting time. They are obtained by subtracting the previous one from each response's timestamp. For each number of users tested, the double node configuration generally yields much smaller time intervals than the other two settings, the larger number, the larger advantage. Even the single edge node tests yield less user waiting time than the centralized setting. As the number of users is increased from 26 to 36, the centralized service suffers from a significant deterioration, resulting in a 14.8% chance for a user to wait for at least 2 seconds and a 5.0% possibility to wait to 3 seconds or more, while the numbers of the single edge setting are 8.7% and 1.6% respectively, and less than 0.5% for the double edge setting. The average, minimal and maximal CPU usages are shown in Fig. 9. The 6-user group scores 193.7%, 201.2% and 124.3% for the single, double edge node and central server respectively. Starting from the 16-user group, where both the single-edge and the centralized settings reach the highest 266.5% and 263.8%, the more concurrent users, the lower CPU usage, and the centralized server decreases more rapidly. The CPU usage of the two-node setting, on the other hand, increases at first and starts decreasing from 26 users.

Analysis: Empirically, the higher user waiting time, the worse the user experience. The experiment results illustrate that as the number of concurrent users increases,

the possibility for a user to wait for a certain long period of time like 3 seconds for the next response is also higher. This deteriorates more rapidly with the centralized configuration. The plausible reasons for this are: i) insufficient computing power; and ii) packet loss in the network. However, by measuring the CPU usage of each experiment, we find that the CPU usage starts decreasing at some point as the number of users increases. If the bottleneck were the CPU, then its usage would have at least maintained at the same level with more users, instead of showing a corresponding decrease. Thus, the loss of packets carrying images and request messages is probably to blame. This reasonably explains both the fact that the response intervals tend to be longer with more users, and that the centralized service tends to be worse than the edge settings. Since each user sends UDP packets at approximately the same rate, the more concurrent users, the higher chance for congestion on each link, which can lead to higher percentage of requests dropped and fewer responses received, contributing to higher intervals. The centralized configuration has an extra hop, the switch, between the user and the Wi-Fi router, thus the possibility of packet loss is even higher. This might have contributed to the disadvantage of the centralized configuration compared to the edge configuration.

One would probably argue that a local server is not a typical configuration nowadays, because many services are deployed on the cloud platforms like Amazon AWS and Microsoft Azure. Our experiment, however, has demonstrated that even in the local network the centralized server does not perform better than the edge. Since there are more hops between the cloud platform and the user, most of which in the uncontrollable internet, it is safe to speculate that a server on the cloud cannot yield better results than the local server in our experiment.

It is obvious from the results that, with two edge nodes, the user experience deteriorates more gradually, and thus can accommodate more users given the same responsiveness standard. This result is based on the assumption that the users are evenly divided to the two edge nodes. If such a division is biased, the performance of the two edge node setting can be somewhat worse. Nevertheless, even with only one edge node to bear all the workload, which is the worst case, the edge computing architecture still defeats the centralized server by a small margin. Therefore, we can safely conclude that the edge computing paradigm that we adopt has made the system more scalable to high concurrency than the traditional centralized server architecture.

C. MAP SIZE

In section II, we argue that our the size of the SLAM map can be too large for smartphones. During the implementation and evaluation process, we constructed a few SLAM maps in different indoor environments. Therefore, we did not specially conduct an experiment for this. Table 2 lists the information of these maps, including their sizes. It can be discovered that the map size is not strictly positively correlated with the space size. This can be largely affected by the number of features

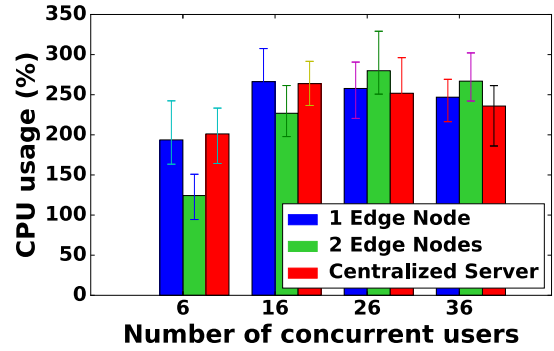


FIGURE 9. CPU usages in each experiment configuration.

TABLE 2. The sizes of map files.

Scenario	Dimensions (m^2)	Map file size (MB)
Local airport terminal	121×33	992
University student center	42×37	518
Department office floor	17×13	287
Network laboratory	8×5	313

and keyframes saved in the map, which in turn depend on both the features available in the environment texture and the fineness of the construction. However, we have shown that, with current techniques, the size of the map file can approach 1GB. This makes it unrealistic to download the map on-the-fly to the mobile devices because of both the time cost and that some devices may not have sufficient space of storage. Although the map size can probably be reduced with more advanced compression techniques and serialization techniques, this would be out of the scope of this paper. With the overly large map size, associated with the previously evaluated advantage in scalability over the centralized server, we are confident that the edge computing paradigm is the optimal choice available.

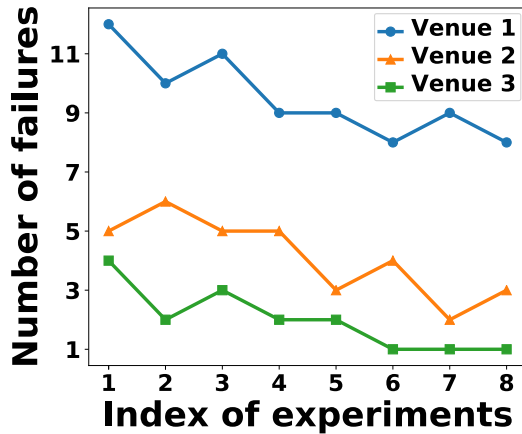
D. AUTOMATIC MAP MAINTENANCE

In this part of the evaluation, we verify the effectiveness of the proposed automatic map maintenance functionality. We treat the system as a blackbox, and test it by walking through a fixed route multiple times and see whether the failure is reduced.

Setup: We choose three indoor spaces for this experiment, whose basic information is listed in Table 3. We use only one edge node to cover the space. In terms of the object detector, we leverage the off-the-shelf Tensorflow Object Detection API [14] and the provided SSD model with ResNet 50 and FPN feature extractors well-trained using the MSCOCO dataset. From the output classes we find whether our interested ones (dynamic objects) exist. To initialize, we construct a relatively rough map along the route, then we carry the smartphone, a Google Pixel 2, and walk through the route.

TABLE 3. Information of experiment venues.

Venue	Dimensions (m^2)	Route length (m)
1	42×37	72
2	17×13	23
3	8×5	13

**FIGURE 10.** The result of automatic map maintenance.

On localization failure, we skip the current position and continue where the localization is successful again. After each round, if there is an update to the map, we wait until the cloud server pushes the updated map to the edge node.

Result: The experiment outcome is shown in Fig. 10. It can be found that, despite the fluctuation, the numbers of failed localization in all three scenarios decrease as the same route is repeated. The number decreases more rapidly at early times than later. Eventually the failure stabilizes around a certain amount.

Analysis: This experiment has verified that our map maintenance mechanism is effective in reducing the localization failures by resulting in a decreasing trend in all scenarios that we tested. However, there is a limitation with this mechanism. The fluctuations shown in Fig. 10 might be a result of random events such as the dynamic objects and the fact that the smartphone is carried by a human, thus the consistency of motion cannot be strictly guaranteed. For example, turning too fast is very likely to lead to ORB-SLAM losing track of the features. The diminishing margin of the decrease in failures, on the other hand, can be due to some particular perspective that has too few features for SLAM to track. The mitigation to this problem is not in the scope of this paper, and we leave it to another research.

E. CASE STUDIES: DYNAMIC ENVIRONMENT

One of the most important challenge for vision-based localization and navigation system is the dynamic environment. Based on our observation, the light condition and dynamic objects affect detection accuracy most. In this section, we conducted two case studies to evaluate the system performance under dynamic environment and explore potential directions to deal with complex environments.

1) SETUP

When dynamic objects exist and light condition varies, there are unexpected number of unrecognized frames receiving from mobile sides. To study how these frames affect the system performance, we designed two cases to mimic the situation. In Fig. 11, we collected data at the same office environment under different light conditions. As shown in Fig. 12, we tested the system when there are individuals sitting in chairs, passing by and no individuals at all. With these two case studies, we diagnose the reasons that lead to failures, and develop corresponding techniques to improve the overall system performance.

2) LIGHT CONDITIONS

For the light condition reason, we apply two methods, which are both aiming to expand image data set and augment the covered scenarios. First, the server generate extra frames in possible light condition, get features from these extra frames, and append these features to the original mapping database. Second, according to the locations where have large chance to loose recognition ability, we relatively supplement data set of specific location in different light conditions when building the map first time.

3) DYNAMIC OBJECTS

In the traditional SLAM, all features of the frames are analyzed and saved while building the map. However, sometimes there are many moving objects in some scenarios, like the crowded corridors, stairs, and long queues. The non relevant objects would occupy too much context. This makes the feature matrix contains too many noise data. As is shown in Fig. 12(d), only background environment are reserved with removing people and bags. This makes our system accurate by avoiding the noise from some useless features, which are non relevant to localization. Besides, some features of dynamic objects can be used when building the map as well as when performing localization, we leave this to future work.

4) ANALYSIS

Vivid is aiming for indoor localization for public areas, like airport, sport filed, office building and etc.. So we include around 72 specific kinds of objects, which are appearing commonly, as our target detecting objects. We choose DetectNet [45] as the object detection model since it can deal with input image of varying sizes and provide reliable accuracies. We use both basic frame data via creating map process and increasing frame data from users when navigating as training data set. We selected 5 different indoor scenarios, including airport, university department building, shopping mall, government building and large hospital. For each scenario, we keep navigating for 30 minutes. The failures occasionally happen, ranging from 8 times to 52 times, after initially mapping. Then the reason of failures are addressed and However, there may be other reasons, like non relevant objects occupying too much context or main features being

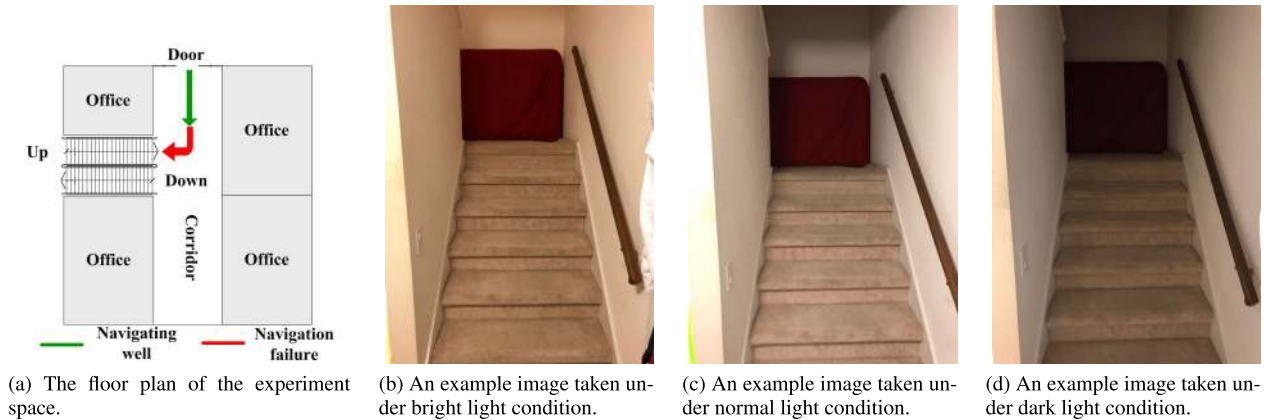


FIGURE 11. Images taken under different light conditions.

covered by dynamic objects. These noise could be conquered via increasing the scenarios when mapping. We leave this to future work.

F. PRIVACY

The user privacy is the top priority in the design of our system. Actually, privacy security is one of the advantages of edge computing. By design, most of the user-uploaded images are processed within the edge node. As the buffer (section III-E) is updated, the outdated images are destroyed. On user disconnection, if no map maintenance is needed, all the data related to the user is deleted. The image sequences for map maintenance, which account for only a small portion of all user images, are uploaded to the cloud server anonymously. The communication safety can be guaranteed by the Transport Layer Security (TLS) techniques. In the cloud server, no record will be kept as to the objects detected. In particular, when human is detected in the frame, the content within the bounding box is immediately removed, as is shown in Fig. 2, no matter this is need by other modules or not. Although the potential of the system may be compromised due to the loss of information, we believe this is worthwhile. The content eventually saved such as features or intensities saved in SLAM have no real-world meaning. Certainly, though, in addition to our consideration, more techniques can be applied and more research on privacy protection can be carried out in the future.

V. DISCUSSION

A. LIMITATIONS

Although the results reported in section IV are encouraging, the experiments are preliminary. The scalability evaluation, for example, compares both the optimal case and the worst case against the centralized server. Luckily, our architecture already prevails even with the worst case. However, it must be pointed out that both cases are not very likely to appear in real-life scenarios, and more edge nodes will probably be used. Therefore, the general superiority of Vivid system needs to be approved with sufficient experiments in real-life scenarios.

One of the challenges that we have not been able to tackle is the dynamic objects. These objects, such as people, can block the camera from the real environment texture, hence make it harder for SLAM to match the features or intensities. Such loss of information has posed significant difficulty to solving this problem, for which there has been no ideal solution so far. Thus, we admit that with the current SLAM technology, our system may not work well in spaces where the population is too dense.

B. FUTURE WORK

In section III-B, we assume the floor is flat and design the grid map to be two-dimensional, which means that there cannot be significant change in height of the terrain. Although currently, the multi-floor support can be achieved with a similar split-splice approach as section III-D, the places where the elevation changes (3D space changes involved), such as the escalators, can hardly be supported. Therefore, the advanced approach of adapting the SLAM map for a complex 3D environmental navigation purpose is the following research focus.

Although the discrete directions that we defined in section III-C has met the need of our navigation system, more refined solutions can be proposed. For example, if we keep the three-dimensional continuous camera pose and the pose is calibrated with the floor plane, then augmented reality (AR) features can be added. This will enable us to display the directions according to the scene (such as printing the instruction arrow along the way) and mark the destination in the UI, providing a better user experience. This would be a useful, fancy but feasible feature to add.

VI. RELATED WORK

A. INDOOR LOCALIZATION AND NAVIGATION

The currently popular solutions to the indoor positioning problem are mostly based on the wireless technology. Among these solutions, ultra wideband (UWB), GPS, Wi-Fi, Bluetooth and their combinations are the most widely adopted approaches. UWB is believed to be more accurate and lower in energy consumption [15], claimed to be able to

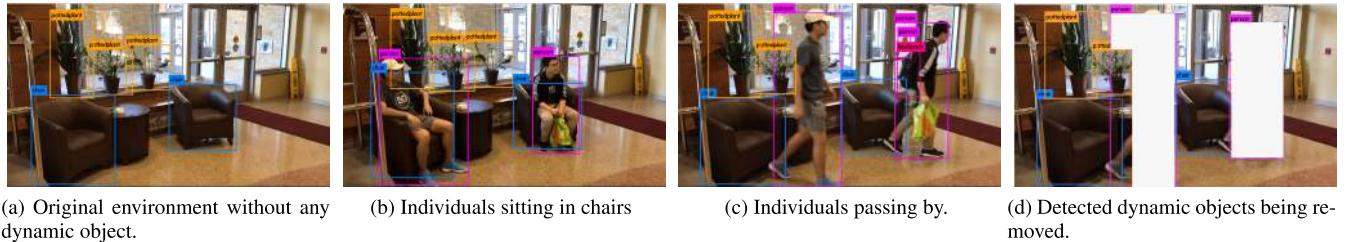


FIGURE 12. Dynamic objects in an example environment.

achieve an accuracy of 20cm [16] with commercial products like Ubisense [17]. However, this technique is generally not available on commodity devices for the public. The GPS estimation of indoor location is normally poor due to the poor signal [16]. COIN-GPS uses a large antenna and cloud computing to solve this problem [1] with a median error of 9.6m. It is neither accurate nor suitable for smart devices due to the extra hardware. The WiFi-based solutions, such as Horus, can achieve 2m accuracy [16], [18], [19], while Chintalapudi et al. proposes a solution that mitigates the need for pre-deployment efforts under the same accuracy [3]. LocBLE is a Bluetooth-based approaches, can reach 1.5m [2]. Empirically, an error under 2m is generally not too difficult for a human brain to correct, but the accumulative error in the process of navigation could be considerable. Therefore, Vivid leveraged vision-based localization techniques to achieve a better accuracy.

The early efforts for human indoor navigation include Cyberguide [20], eGuide [21] and the Resource-Adaptive Mobile Navigation System [22]. These systems use remote controls as infrared beacons, which are too expensive to scale [23]. Later on, inertial sensors are applied to track the user motion [24], which needs the user to constantly calibrate the position, which can be annoying in large spaces. Hile and Borriello's work uses scale-invariant feature transform (SIFT) to estimate the pose. Despite that it relies on a server for the heavy calculation, an excessively long processing time of 10 seconds per image is reported [25]. In recent years, the fusion of different sensors has become the trend. FollowMe [26] gathers data from the motion sensors and barometer, to determine the location and direction by comparing the features of the data with the trace left by a leader. TraviNavi [27] combines the Wi-Fi, IMU, and visual recognition techniques. Both solutions have a median error under 2m, while they the same drawback since both also rely on the quality of traces, which is not trivial to control.

B. SLAM

SLAM is a technique that solves the problem of obtaining the 3D structure of the environment and the motion of the sensor [28]. The sensors that can be leveraged include light detection and ranging (LIDAR) [29], the inertial measurement unit (IMU), GPS, and cameras [40], [46]. The visual SLAM (vSLAM) is a branch of SLAM that uses visual information only, such as images and depth. vSLAM is actively researched in computer vision and robotics, with three major

types of input information: monocular, stereo, and RGB-D. There are two approaches to solving this problem, featured-based and direct. Feature-based solutions extract the features from the image input, and estimates the motion of the camera from the displacement of the features between successive images, represented by MonoSLAM [30], PTAM [31] and ORB-SLAM [6]. On the other hand, DTAM [32] and LSD-SLAM [9] adopt the direct method that does not abstract from the images, but uses them as a whole. The most recent proposal ORB-SLAM [6] achieves merely 1% error of the map dimensions.

However, the current monocular SLAM solutions are not suitable for running on smart devices for our purpose. DTAM [32] needs GPU to run in real time, LSD-SLAM [9] and ORB-SLAM [6] work on computer CPU. There is a PTAM system proposed for mobile phones, claiming to be real-time if the map is small [33], but the specific scale is not mentioned. In our work, we are focusing design an indoor navigation system that can work in a large space which would be more than 100m in length and width. Therefore, it is very likely that the current SLAM systems will be heavy burdens on both the CPU and the battery of smart devices. Hence, Vivid is designed to offload computations to the edge computing platform.

C. EDGE COMPUTING

Edge computing refers to the concept that computation is performed at the edge of the network [34]. Traditionally, computation tasks are executed on the centralized cloud servers (although they might be distributed inside). With the boom of IoT devices, an increasingly large amount of data is produced and the burden to transport the data and to process the data is more and more heavy. Having computation jobs accomplished at the edge reduces the traffic in the network and keeps data local, which contributes to lower latency, higher bandwidth and higher privacy [34], [35]. ParaDrop [7], [8], for example, is a project that enables Wi-Fi routers to act as edge nodes, on which run third-party services that talk to the users directly connected to it. Remote control is also allowed to manage and coordinate the edge nodes.

Although many investigations into the concept and perspective of edge computing has been carried out [34]–[37], [44], [47], the specific applications that apply this paradigm are rare, the examples including a mobile edge computing-based video streaming technique [38] proposed by Trans et al., and Kumar et al. leveraging edge computing to

handle large data sets in smart electric grids [39]. Our work, benefitting much from the edge computing architecture, is yet another concrete application contributed to this realm.

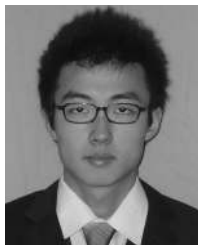
VII. CONCLUSION

In this paper, we have presented Vivid, a vision-based indoor navigation system for commodity smart devices. The system adopts a cloud-assisted edge computing architecture to distribute the network throughput and offload the computation tasks from a centralized cloud server to edge nodes. We also propose a grid-based discrete map to eliminate the need for the domain knowledge on the environment, and a deep learning-based automatic map maintenance mechanism to mitigate human involvement in map updates due to environmental changes. The evaluation results show that: i) Vivid's localization error is at most 60cm, lower than most, if not all existing works; ii) our architecture provides better scalability compared to centralized service; iii) the automatic map maintenance is effective in response to unrecognized environmental context.

REFERENCES

- [1] S. Nirjon, J. Liu, G. DeJean, B. Priyantha, Y. Jin, and T. Hart, "COIN-GPS: Indoor localization from direct GPS receiving," in *Proc. 12th Annu. Int. Conf. Mobile Syst., Appl., Services*, 2014, pp. 301–314.
- [2] D. Chen, K. G. Shin, Y. Jiang, and K.-H. Kim, "Locating and tracking BLE beacons with smartphones," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.*, 2017, pp. 263–275.
- [3] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan, "Indoor localization without the pain," in *Proc. 16th Annu. Int. Conf. Mobile Comput. Netw.*, 2010, pp. 173–184.
- [4] R. Faragher and R. Harle, "Location fingerprinting with Bluetooth low energy beacons," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 11, pp. 2418–2428, May 2015.
- [5] C. Yang and H.-R. Shao, "WiFi-based indoor positioning," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 150–157, Mar. 2015.
- [6] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Aug. 2015.
- [7] D. Willis, A. Dasgupta, and S. Banerjee, "ParaDrop: A multi-tenant platform to dynamically install third party services on wireless gateways," in *Proc. 9th ACM Workshop Mobility Evolving Internet Archit.*, 2014, pp. 43–48.
- [8] P. Liu, D. Willis, and S. Banerjee, "ParaDrop: Enabling lightweight multi-tenancy at the Network's extreme edge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 1–13.
- [9] J. Engel, T. Schops, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 834–849.
- [10] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.
- [11] Alkaid-Benetnash. (2018). *Map Save Load and Bin Vocabulary*. [Online]. Available: https://github.com/raulmur/ORB_SLAM2/pull/381
- [12] R. Ramey. (2008). *Boost Serialization Library*. [Online]. Available: www.boost.org/doc/libs/release/libs/serialization
- [13] Open CV. (2018). *Camera Calibration*. [Online]. Available: <https://docs.opencv.org/3.4/dc/dbb/tutorialpycalibration.html>
- [14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/Accuracy trade-offs for modern convolutional object detectors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, vol. 4.
- [15] A. Alarifi, A. Al-Salman, M. Alsaleh, A. Alnafessah, S. Al-Hadhrani, M. Al-Ammar, and H. Al-Khalifa, "Ultra wideband indoor positioning technologies: Analysis and recent advances," *Sensors*, vol. 16, no. 5, p. 707, May 2016.
- [16] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 37, no. 6, pp. 1067–1080, Nov. 2007.
- [17] T. U. Company. (2018). *Ubisense*. [Online]. Available: <http://www.ubisense.net/>
- [18] M. A. Youssef, A. Agrawala, and A. U. Shankar, "WLAN location determination via clustering and probability distributions," in *Proc. 1st IEEE Int. Conf. Pervasive Comput. Commun.*, Mar. 2003, pp. 143–150.
- [19] M. Youssef and A. Agrawala, "Handling samples correlation in the horus system," in *Proc. IEEE INFOCOM*, vol. 2, Mar. 2004, pp. 1023–1031.
- [20] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: A mobile context-aware tour guide," *Wireless Netw.*, vol. 3, no. 5, pp. 421–433, 1997.
- [21] T. Kirste and J. L. Encarnacao, "Beyond the desktop: Natural interaction and intelligent assistance for the everyday life," *CG Topics*, vol. 3, pp. 16–19, 2000.
- [22] J. Baus, A. Kruger, and W. Wahlster, "A resource-adaptive mobile navigation system," in *Proc. 7th Int. Conf. Intell. Interfaces*, 2002, pp. 15–22.
- [23] G. N. Desouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 2, pp. 237–267, Feb. 2002.
- [24] D. Merico and R. Bisiani, "Indoor navigation with minimal infrastructure," in *Proc. 4th Workshop Positioning, Navigat. Commun.*, Mar. 2007, pp. 141–144.
- [25] H. Hile and G. Borriello, "Information overlay for camera phones in indoor environments," in *Proc. Int. Symp. Location-Context-Awareness*. Berlin, Germany: Springer, 2007, pp. 68–84.
- [26] Y. Shu, K. G. Shin, T. He, and J. Chen, "Last-mile navigation using smartphones," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 512–524.
- [27] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, and F. Zhao, "Travi-navi: Self-deployable indoor navigation system," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2655–2669, Oct. 2017.
- [28] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, no. 1, p. 16, Jun. 2017.
- [29] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 1271–1278.
- [30] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.
- [31] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. 6th IEEE ACM Int. Symp. Mixed Augmented Reality*, Nov. 2007, pp. 225–234.
- [32] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2320–2327.
- [33] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *Proc. 8th IEEE Int. Symp. Mixed Augmented Reality*, Oct. 2009, pp. 83–86.
- [34] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
- [35] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
- [36] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, 2015, pp. 37–42.
- [37] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," 2015, *arXiv:1502.01815*. [Online]. Available: <http://arxiv.org/abs/1502.01815>
- [38] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," in *Proc. 13th Annu. Conf. Wireless Demand Netw. Syst. Services (WONS)*, Feb. 2017, pp. 165–172.
- [39] N. Kumar, S. Zeadally, and J. J. P. C. Rodrigues, "Vehicular delay-tolerant networks for smart grid data management using mobile edge computing," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 60–66, Oct. 2016.
- [40] B. Qi, W. Zhao, X. Wang, S. Li, and T. Runge, "A low-cost driver and passenger activity detection system based on deep learning and multiple sensor fusion," in *Proc. 5th Int. Conf. Transp. Inf. Saf. (ICTIS)*, Jul. 2019, pp. 170–176.

- [41] S. E. Crouter, P. L. Schneider, M. Karabulut, and D. R. Bassett, "Validity of 10 electronic pedometers for measuring steps, distance, and energy cost," *Med. Sci. Sports Exerc.*, vol. 35, no. 8, pp. 1455–1460, Aug. 2003.
- [42] D. R. Bassett, B. E. Ainsworth, S. R. Leggett, C. A. Mathien, J. A. Main, D. C. Hunter, and G. E. Duncan, "Accuracy of five electronic pedometers for measuring distance walked," *Med. Sci. Exerc.*, vol. 28, no. 8, pp. 1071–1077, Aug. 1996.
- [43] J. Stalbaum and J.-B. Song, "Keyframe and inlier selection for visual SLAM," in *Proc. 10th Int. Conf. Ubiquitous Robots Ambient Intell. (URAI)*, Oct. 2013, pp. 391–396.
- [44] B. Qi, P. Liu, T. Ji, W. Zhao, and S. Banerjee, "DrivAid: Augmenting driving analytics with multi-modal information," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Dec. 2018, pp. 1–8.
- [45] J. Barker, S. Sarathy, and T. Andrew. (Jul. 2016). *DetectNet: Deep Neural Network for Object Detection in DIGITS*. Nvidia. [Online]. Available: <https://devblogs.nvidia.com/parallelorall/detectnet-deep-neural-network-object-detection-digits>
- [46] W. Zhao, J. Yin, X. Wang, J. Hu, B. Qi, and T. Runge, "Real-time vehicle motion detection and motion altering for connected vehicle: Algorithm design and practical applications," *Sensors*, vol. 19, no. 19, p. 4108, Sep. 2019.
- [47] P. Liu, B. Qi, and S. Banerjee, "EdgeEye: An edge service framework for real-time intelligent video analytics," in *Proc. 1st Int. Workshop Edge Syst., Analytics Netw.*, 2018, pp. 1–6.



WEI ZHAO received the master's degree in transportation planning and management from Dalian Maritime University and the Ph.D. degree from the University of Wisconsin–Madison, Madison, WI, USA, where he is also pursuing another Ph.D. degree with the College of Agricultural and Life Sciences. His research interests include connected and automated vehicles, computer vision applied in traffic analysis, and AI farm.



LIANGJIE XU received the B.S. and M.S. degrees from the Wuhan University of Science and Technology, in 1989 and 2000, respectively, and the Ph.D. degree in transportation planning and management from Southeast University, in 2005. She works as a Professor with the Department of Traffic Engineering, Wuhan University of Technology. Her research interests include traffic optimization and urban traffic information induction control.



BOZHAO QI received the B.S. degree in electrical engineering and computer science from Case Western Reserve University and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Wisconsin–Madison. He is a Research Assistant with the Department of Computer Science, University of Wisconsin–Madison. He is a member of the WiNGs Lab, Department of Computer Science, under the supervision of Prof. S. Banerjee. His research interests include the fields of mobile computing, mobile health, context awareness, and ubiquitous computing. He has worked on several vehicular-related projects during the Ph.D. degree study. The topics of projects cover sensing vehicle dynamics, transit analytics, human mobilities, and so on. He is currently working on the driving behavior detection and evaluation.



JIA HU received the B.S. degree in civil engineering from Zhejiang University, the master's degree in transportation engineering from North Carolina State University, and the Ph.D. degree from the University of Virginia. He works as a Hundred Talent Program Professor with the College of Transportation Engineering, Tongji University. Before joining Tongji University, he was a Research Associate with the Federal Highway Administration, USA (FHWA). His research interests include connected and automated vehicles, microscopic simulation model applications, system optimization, and transportation energy efficiency. He is also Chair of Vehicle Automation and Connectivity Committee of the World Transport Convention. Furthermore, he is a member of TRB (a division of the National Academies) Vehicle Highway Automation Committee and the Simulation Subcommittee of Traffic Signal Systems Committee and a member of the Advanced Technologies Committee of the ASCE Transportation and Development Institute. He is an Associate Editor of the *American Society of Civil Engineers Journal of Transportation Engineering* and an Editorial Board Member of the *International Journal of Transportation*.



TENG WANG received the B.S. and M.S. degrees in civil engineering from Iowa State University and the Ph.D. degree in civil engineering from the University of Kentucky. He is an Assistant Research Scientist with the Texas A&M Transportation Institute (TTI), Texas A&M University System. Prior to joining TTI, he worked as a Post-doctoral Associate with the University of Kentucky and the Alabama Transportation Institute on transportation safety and policy related studies. He is a registered Professional Engineer (PE) in the State of Kentucky (Transportation). His professional and research interests include transportation policy and legislation, relationship between road features and safety, statistical assessment of safety issues, applications of geographic information systems (GIS) to transportation, remote sensing and image analysis using GIS, and 3D infrastructure condition assessment. He serves as a Young Professional Member of the Transportation Research Board (TRB) Highway/Rail Grade Crossings Committee (AHB60) and the Co-Chair of New Technologies and Applications Committee of World Transport Convention (WTC). He also serves as the paper Reviewer for several civil engineering journals such as *Computer-Aided Civil and Infrastructure Engineering (CACAE)*, *Transportation Research Part C*, and *Earthquake Engineering and Engineering Vibration (EEEV)*.



TROY RUNGE received the B.S. degree from UW-Stevens Point and the M.S. and Ph.D. degrees from the Institute of Paper Science and Technology, Georgia Institute of Technology. He is an Assistant Professor in biological systems engineering in CALS, where he performs research and teaches in the bioenergy field. He is a Lignocellulose Chemist by training and has Pulp and Paper Science degrees. He spent several years at UW as the Director of the Wisconsin Bioenergy Initiative, and prior to that, he spent fifteen years working at Kimberly-Clark Corporation in a variety of research and engineering roles for pulp, tissue, nonwoven, and hygiene product production. He is currently working in several aspects of bioenergy and bio-based materials with an emphasis on biomass composition and separation technologies.

...