Image-Based Indoor Topological Navigation with Collision Avoidance for Resource-Constrained Mobile Robots

Suman Raj Bista · Belinda Ward · Peter Corke

Received: 10 July 2020 / Accepted: 1 April 2021 / Published online: 3 June 2021
© The Author(s), under exclusive licence to Springer Nature B.V. 2021

Abstract

This paper presents a complete topological navigation system for a resource-constrained mobile robot like Pepper, based on image memory and the teach-and-repeat paradigm. Image memory is constructed from a set of reference images that are acquired during a prior mapping phase and arranged topologically. A* search is used to find the optimal path between the current location and the destination. The images from the robot’s RGB camera are used to localize within the topological graph, and an Image-Based Visual Servoing (IBVS) control scheme drives the robot to the next node in the graph. Depth images update a local egocentric occupancy grid, and another IBVS controller navigates local free-space. The output of the two IBVS controllers is fused to form the final control command for the robot. We demonstrate real-time navigation for the Pepper robot in an indoor open-plan office environment without the need for accurate mapping and localization. Our core navigation module can run completely onboard the robot (which has quite limited computing capabilities) at 5 Hz without requiring any external computing resources. We have successfully performed navigation trials over 15 days, visiting more than 50 destinations and traveling more than 1200m with a success rate of over 80%. We discuss remaining challenges and openly share our software.

Keywords Visual navigation · Visual servoing · Collision avoidance · Pepper robot

1 Introduction

Navigation is a fundamental problem in mobile robotics. For a mobile robot to be autonomous, it should be able to navigate in the environment on its own. For navigation to be successful and robust, the robot must also be able to avoid collision. Vision has become a mainstream sensor for mobile robot navigation [6, 19]. Classical approaches to the visual navigation problem rely on the geometry of the environment and use metric techniques such as 3D pose estimation as in simultaneous localization and mapping (SLAM) methods [12, 20, 38]. Navigation then involves regulation of the error between the current localization and the specified trajectory that the robot has to follow in metric space. An alternative approach is to use appearance-based techniques that work directly in the sensor space and do not require explicit representation of the environment in the metric space. In such navigation, the error signal is measured directly in the images and mapped to actuator commands. The environment in such cases is represented topologically [4, 11, 14, 25, 30].

This paper is focused on navigation for robots like Pepper (Fig. 1a), which do not have sophisticated sensors or sufficient onboard computing power to perform accurate metric localization and navigation. The work of [33] has demonstrated indoor navigation with the Pepper robot by fusing data from the robot’s point lasers and depth camera, but that method still requires a 2D metric map and external computational resources. Simple image-based navigation for the Pepper was demonstrated in [3] using point and line features, but the computation was done remotely and the navigation lacks collision avoidance. Our goal is to
perform reliable and robust navigation of the Pepper robot with just onboard processing using the image memory that the robot has obtained during its supervised (human guided) teaching phase as shown in Fig. 4. Image memory consists of reference images organized in a directed graph. Navigation is performed based on these reference images and can be considered a form of visual teach-and-repeat [10, 13]. Works in visual teach-and-repeat like [3–5, 11, 14, 17, 25, 30, 36] have shown that accurate mapping and localization are not mandatory for visual navigation. Direct methods like [5, 11, 25] compare entire images between teach and repeat runs, but they are less robust to large image shifts and appearance change in the scene. The recent deep learning-based teach-and-repeat navigation such as [7, 34] are not suitable to run onboard in the resource-constrained platform like ours. The point-based features have been used in outdoor navigation such as [14, 17, 30, 36]. The performance of these methods drops significantly because of windows, wiry structures, reflections and repetitions, and limited texture in indoor scenarios, which causes point-based features to perform poorly indoor [26]. Furthermore, these methods are computationally intensive too. The methods such as [4, 20, 31, 38] showed that line segments are good features for indoor navigation. Furthermore, line segments are more robust to moderate appearance changes, occlusions and motion blur than the points. We have chosen [4] as our starting point because it is robust enough for the indoor environment while also being sufficiently computationally light-weight to run onboard. However, [4] lacks obstacle avoidance and thus does not guarantee that the robot will always move with appropriate clearance from obstacles such as walls and furniture. This capability is essential for the safety of the robot, and for any humans in the vicinity. The robot frequently got stuck when it moved through the narrow corridors using only [4]. Since this navigation method does not rely on a global 3D model of the environment, a compatible obstacle avoidance approach should be model-free and not rely on accurate pose estimation.

The works of [23, 24] are model-free obstacle avoidance methods, but they still rely on accurate pose estimation of the robot. Redundancy-based schemes to avoid obstacles are presented in [9, 15], [15] have used ultrasound, while [9] have used a laser to detect the obstacles and control the pan-angle of the camera to maintain visibility of the features. These methods are able to avoid static and moving obstacles during navigation. However, for robots like Pepper with sensors in the head, which should face forward in the direction of motion for safety, the pan angle cannot be controlled. Velocity space methods like the Dynamic Window Approach (DWA) by [16] and its variants such as [2, 29] use constrained optimization in the velocity space to avoid the obstacles and generate smooth trajectory. However, DWA requires careful parameter tuning and may fail to travel through narrow spaces [27]. A complete humanoid navigation scheme based on image memory with obstacle avoidance has been proposed for the Nao robot by [13]. Their obstacle avoidance is used for path planning, peripheral avoidance of unexpected static objects, and to step over small obstacles. In our case, the wheeled Pepper robot is constrained to drive around, rather than step over, obstacles.

Our contribution is a complete method for indoor navigation (mapping, path planning, localization using a topological graph, collision avoidance and motion control) of a mobile robot like Pepper, which has limited onboard computing power, and low cost and average quality navigation sensors. The previous work [4] consist of image-based navigation only along a single sequence of reference images. There was no collision avoidance and path planning over a topological graph. We extend the previous work to include obstacle avoidance and topological path planning in a computationally efficient manner, which can run on the Pepper robot. First, we extend the earlier image-based
navigation to work with a topological graph. Then, we develop a lightweight free-space navigation using the depth camera. Finally, we propose the fusion of control commands from the image-based navigation module and the free-space navigation module that ensures the robot moves freely to the destination without collision or becoming stuck along the path. Navigation does not require a metric map, only a sequence of RGB images arranged in a topological graph. The path planning is performed using A* search [21] over the topological graph. The navigation task is the fusion of goal-directed navigation based on image memory, and collision-free navigation based on the depth camera. We have demonstrated this approach, running completely onboard without external processing, on the wheeled humanoid Pepper robot. We have demonstrated operations over 15 days and provide detailed performance data. The complete software is open-sourced.1

The next section describes the complete framework for mapping and navigation. The key details from the previous work [4] are also included for completeness. Section 3 presents some experimental results in different indoor scenarios, which demonstrate the validity of the proposed navigation scheme. Finally, some concluding remarks and remaining challenges are reported in Section 4.

2 Mapping and Navigation Framework

Pepper (Fig. 1a) is a popular humanoid robot developed by [32]. It is equipped with two RGB cameras and one depth camera in its head, and an omni-directional base that enables it to move freely in any direction. Each RGB camera provides a resolution up to 2560 × 1920 at 1 frame per second (fps) or 640×480 at 30 fps. We use the upper camera because it looks ahead rather than at the floor (Fig. 1b). The robot also has three laser rangefinders located at the base of the robot as shown in Fig. 2a. Each laser sensor provides only 15 points across its horizontal field-of-view (FOV) of 60°, and the sensors are non-overlapping. All of these factors make the laser sensors unsuitable as the sole sensor for obstacle avoidance or SLAM. However, the robot is equipped with one ASUS Xtion depth sensor (Fig. 2b) that is located behind its eyes. It provides image resolution up to 320 × 240 at 20 fps and has a horizontal FOV of 58° and vertical FOV of 45° with a working range from 40cm to 8m. The depth sensor data is noisy, but we have shown it is sufficient for our purpose of avoiding collision. The robot has a 1.91 GHz ATOM E3845 processor and 4 GB RAM [32], which is shared with its internal processes. In summary, the Pepper robot has a good mobility base but limited sensing and computational capabilities.

For our navigation purpose, the Pepper robot is considered as a non-holonomic mobile robot of unicycle type equipped with a fixed perspective camera. The intrinsic parameters of the camera are constant and coarsely known. During navigation, it is assumed that the robot is initially inside the mapped environment. In the proposed method, we use the top RGB camera to perform the image-based navigation that defines the path the robot has to follow, while the depth camera is used for local collision avoidance. The control commands from both modules are fused together to derive the final velocity command for the base.

2.1 Mapping

During mapping or teaching (Fig. 3), the robot is moved in the navigation environment under human supervision. The robot is led by the hand as shown in Fig. 4, which we have programmed to act as a joystick. Images of the environment are captured as the robot moves and are stored as a teaching sequence. From this teaching sequence, a small subset of images is selected, which is known as the image memory. This image memory consists of reference images that represent particular locations in the environment. The selected reference images are organized.

---

1The source code is available on https://github.com/qcr/pepper_navigation.git
in an adjacency graph, which gives the topological representation of the environment. The reference images are selected automatically, whereas creating the top-level topological map involves some manual interventions as discussed in Section 2.1.2. The mapping process is shown in Fig. 3.

2.1.1 Reference Images Selection

The reference images are automatically selected such that they are sufficiently different from the adjacent reference images. Differences are determined from a similarity metric, which depends upon local geometric features. We choose line segment features, and the similarity metric is obtained from feature matching/tracking. In indoor environments, line segments are abundant. In addition to this, line segments are more robust to partial occlusions and some change in the illumination, and more resilient to motion blur [26, 38] than points. The tracking of points fails often due to varying illumination level and motion blur. Also, line segments in the image can be detected [1, 35] and matched [37] accurately in real-time. These are the main reasons for motivation to use line segments as visual landmarks in this work.

Let $I^c$ be the currently acquired image, $I^a$ be the image acquired just before $I^c$, and $I^r$ be the most recent reference image. The features detected in $I^r$ are matched over $I^c$ with outlier rejection. If $m$ is a positive integer and $I^r \neq I^a$ a new reference is selected when

$$\tilde{F}_3(I^r, I^a, I^c) < m.$$  (1)

The metric $\tilde{F}_3(\cdots) \in \mathbb{N}$ is the number of features matched between three views as in [3, 4]. We obtain $n$ reference images $I^1, I^2, I^3, \ldots, I^n$, which are the image memory for navigation. In this work, the method proposed in [4] has been used to track lines, which is based on three-view line matching with outlier rejection based on RANSAC and trifocal tensor estimation [22]. More details about reference image selection from the teaching sequence are provided in [4].

Instead of choosing a reference image every fixed distance using odometry, we have used the approach mentioned above to ensure that there are enough reference images in the case of turns and systematic variation in illumination. There is rapid displacement of line segments when turning, whereas variation in illumination may result in poor performance in line segments matching. Our approach ensures that there are at least $m$ matched line segments between the reference images, which is critically important for good localization and smooth control. In all our experiments, we have used $m = 20$. 

Fig. 3 Mapping: Reference images selection and creation of the topological graph

Fig. 4 Acquiring images of the environment for mapping
2.1.2 Creation of Topological Graph

From the obtained image memory, the topological graph is created. The nodes and edges are defined by the user. The nodes are defined based on the higher-level semantics relevant to human users, for example, destination points like someone’s room, robotics lab etc. The edges comprise a sequence of reference images in the adjacency graph. A pair of nodes are connected by two edges: one representing the path in a forward direction and the other representing the path in the opposite direction as shown in Fig. 5. The link between reference images in adjacent edges is done automatically based on feature matching, which is essential for smooth motion when the robot moves from one edge to another.

Let us consider a directed topological graph $G$ that has $p$ nodes and $q$ edges. Let the nodes be represented by $N_1, N_2, ..., N_p$ and the edge connecting start node $N_i$ and end node $N_j$ be represented by $E_{ij}$. The edge $E_{ji}$ represents the travel in the opposite direction with start node $N_j$ and end node $N_i$ (Fig. 5). The cost of the edge is set to be the number of reference images on that edge, which is a proxy for distance.

2.2 Navigation

Navigation involves path planning and localization in the topological graph, collision avoidance and motion control. Based on the robot’s current location in the topological graph and defined destination, path planning outputs the list of reference images in the optimal path (Fig. 6). This list of the sparse reference images represents the reference path for navigation. Once path planning is complete, successive localization, collision avoidance and motion control run concurrently as the robot navigates from start to destination. During navigation, it is assumed that the robot is initially inside the mapped environment.

2.2.1 Localization in the Topological Graph

The very first step in navigation is to localize globally in the topological graph. The current image is compared with the entire image memory to determine the current edge. In this work, we have used the number of matched line segments between the views as a similarity metric [4]. If $S_L(\cdot)$ is a
two-view based similarity metric, then \( I^R_k \) is the best match in the image memory as

\[
k = \arg\max_{p \in \{1\ldots n\}} \left\{ \delta_2 \left( I^c, I^R_p \right) \right\}. \tag{2}
\]

In (2), if there are two or more reference images with an extremely good match with \( I^c \), we can simply check with the next reference image to eliminate the false matches. Such cases may occur in a region where there are few matching line segments. Each image can belong to only one edge. If \( I^R_k \) is localized in the edge \( E_{ij} \), the robot’s initial location is somewhere between start node \( N_i \) and end node \( N_j \). Let \( N_f \) be the node that is at the end of the edge in the direction the robot is facing and \( N_b \) be the node at the end of the edge behind the robot. We define, \( N_f = N_j \) and \( N_b = N_i \), which are used in the next section.

### 2.2.2 Optimum Path Selection in the Graph using A*

Based on the current location and defined destination, the optimal path is selected using A* search on the graph. Let \( N_S \) be the start node at one of the two ends of the current edge and \( N_D \) be the destination node. This destination node \( N_D \) is either selected by human or selected randomly in experiments. We perform two A* searches from \( N_S \) to \( N_D \), first with \( N_S = N_f \) and the second with \( N_S = N_b \). After that, we choose the path with the lowest cost as the optimal path. If the optimal path is for \( N_S = N_b \), the robot is rotated by \( 180^o \) before the actual navigation commences. After that, we prepend the other node (which is at the end of the current edge behind the robot) in the optimum path so as to ensure the robot’s current edge is always in the optimal path. We represent the final list of nodes in the optimum path as \( \mathcal{N} \).

### 2.2.3 Selection of Reference Images in the Optimum Path

The reference image sequence \( \{I^R\} \) for the entire path, defined by the list of nodes \( \mathcal{N} \), is extracted from the image memory. For example, if \( \mathcal{N} = (N_S, N_a, N_b, N_D) \), we extract the reference images from edges \( E_{Sa}, E_{ab}, \) and \( E_{bD} \). Such reference image sequence visually represents the path from source to destination. Let \( I^R \) has \( I \) reference images. Then \( I^R = (I^R_1, I^R_2, \ldots, I^R_L) \), where \( I^R_1, I^R_2, \ldots, I^R_L \) are individual reference images in the optimum path. Once we have \( I^R \), we can perform image-based navigation as in [3, 4, 14].

### 2.2.4 Navigation Towards the Goal

Before the actual navigation commences, the robot is rotated by \( 180^o \) if it is not facing along the direction of the optimal path. This rotation information is obtained from Algorithm 1.

---

**Algorithm 1**: Selection of the optimal path using A* search algorithm.

**INPUTS:**

\( G \): Topological Graph.

\( N_f, N_b \): end nodes of edge containing current position.

\( N_D \): Destination node.

**OUTPUT:**

\( \mathcal{N} \): list of nodes of the optimum path.

\( Rotate180 \): flag to indicate whether robot has to perform initial \( 180^o \) turn.

Let \([P, C]=AstarSearch(G, N_S, N_D)\), where \( P \) is a list of nodes for optimal path from \( N_S \) to \( N_D \) in graph \( G \) and \( C \) is the total path cost.

\([P_f, C_f]=AstarSearch(G, N_f, N_D)\)

\([P_b, C_b]=AstarSearch(G, N_b, N_D)\)

If \( C_f > C_b \)

\( \mathcal{N} = (N_f, P_b) \)

\( Rotate180 = TRUE \)

Else

\( \mathcal{N} = (N_b, P_f) \)

\( Rotate180 = FALSE \)

Note: The node is appended in front of the optimal path so as to make sure the robot’s current edge is always in the optimal path.

If there are only two nodes in the optimal path, \( \mathcal{N} = (N_S, N_D) \).

The navigation task is further divided into three subtasks: image-based navigation, obstacle/collision avoidance, and fusion of control. Figures 7 and 8 presents an overview of navigation towards the goal.

**a) Image-based Navigation** The objective of the image-based navigation is to localize in the reference image sequence and generate the appropriate control command to reach towards the goal. The localization, in this case, is to find the adjacent reference images that best match the current view. In Fig. 9, \( I^c \) is the current view of the robot. The position of \( I^c \) in the graph is between reference images \( I^R_k \) and \( I^R_{k+1} \), where \( I^R_k \) is the previous reference image and \( I^R_{k+1} \) is the next reference image. For smooth control and successive localization, we also use the second-next reference image \( I^R_{k+2} \). The best matches are obtained from feature matching. Correct localization is essential for successful navigation. We need to perform two types of localization:
i) **Global Localization in the Optimum Path** The current view is compared with all the reference images in the optimum path obtained from A* to find the best pair of adjacent reference images. This localization is always performed at the beginning of navigation, and during navigation to recover the robot if it is temporarily lost. The position of \( I^c \) in the optimal path is bracketed by a pair of reference images \( (I_k, I_{k+1}) \), which is obtained from

\[
k = \arg \max_{i \in 1 \cdots l-1} \left\{ \tilde{s}_3 \left( I^R_k, I^c, I^R_{k+1} \right) \right\}.
\]  

(3)

ii) **Successive Localization** This localization takes advantage of the adjacency relationship between the reference images on the optimum path. Once we have global localization, further localizations can be performed by comparing only a few reference images in the neighborhood. The current image \( (I^c) \) is compared with three reference images: \( I^R_k, I^R_{k+1}, I^R_{k+2} \) (Fig. 9). Reference image switching \( (k \leftarrow k+1) \) occurs when \( I^R_{k+1} \) precedes \( I^c \) i.e. when either of two criteria is fulfilled

\[
\tilde{s}_3 \left( I^c, I^R_k, I^R_{k+2} \right) > \tilde{s}_3 \left( I^R_k, I^c, I^R_{k+1} \right), \text{ or}
\]

\[
\tilde{s}_2 \left( I^c, I^R_{k+1} \right) > \tilde{s}_2 \left( I^R_k, I^c \right) \text{ and } \tilde{s}_2 \left( I^R_{k+1}, I^R_{k+2} \right) > \tilde{s}_2 \left( I^c, I^R_k \right).
\]  

(4)

The second criterion is useful when there are few three-view correspondences, which may occur with sharp turns in corridors that have little texture.

The robot is said to be at the goal destination when no further switching of reference images is possible. In other words, navigation stops when \( I^R_{k+1} = I^R_l \), where \( I^R_l \) is the last reference image in the optimum path.

---

**Fig. 7** Global Localization in the optimum path. The output of this localization is the reference images that matched best with the current view.

**Fig. 8** Navigation: Image-based localization in the optimal path (Section 2.2.4), collision avoidance using depth camera (Section 2.2.4) and computation of forward and rotational velocities. The reference image sequence of the optimum path is obtained as discussed in Section 2.2.3.
Feature Selection and Control For image-based navigation, we have used the method described in [4] using line segment features with some minor modifications. We use the Edge Drawing Lines (EDLines) detector [1] and the line matching method proposed by [37] respectively to detect and match the line segments. These algorithms are chosen because of their high accuracy and fast computation that is sufficient for reliable real time navigation.

Figure 10 shows an instance of line segments matching in three views. The selection of the reference images, localization on the topological graph and control of the robot is based on the matching of the line segments as discussed in [4]. We have selected this method rather than [3] for computational reasons. On the Pepper platform, [4] is able to run at 8Hz, while [3] can only run at 2 Hz.

During navigation, the robot qualitatively follows the taught path defined by the reference image sequence. The forward velocity is kept constant and reduced to a smaller value when turning. The rotational velocity is derived using the reference images and the current image via an image-based visual-servoing (IBVS) control law [8] described below.

Let us define a vector of visual features (line segments) as \( s \), the camera velocity expressed in camera frame as

\[
\mathbf{v}_c = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)^T,
\]

where \( v \) is the linear velocity and \( \omega \) is the rotational velocity around the given axes. The velocity of \( s \) can be related via an interaction matrix \( \mathbf{J}_s \) [8] to \( \mathbf{u}_c \) as

\[
\dot{s} = \mathbf{J}_s \mathbf{u}_c.
\]

For the considered unicycle-like robot (Fig. 11a), \( \mathbf{u}_c \) can be expressed in terms of robot velocity \( \mathbf{u} = (v_r, \omega_r)^T \) as

\[
\mathbf{u}_c = (-\delta \omega_r, 0, v_r, 0, -\omega_r, 0)^T,
\]

where \( \delta \) is the distance between the camera center and the robot center of rotation, \( v_r \) is the forward velocity and \( \omega_r \) is the rotational velocity of the robot. Now, from (6) and (7), we obtain

\[
\dot{s} = \mathbf{J}_e v_r + \mathbf{J}_\omega \omega_r.
\]

where \( \mathbf{J}_e \) and \( \mathbf{J}_\omega \) are the Jacobians associated with \( v_r \) and \( \omega_r \) respectively. In practice, \( \mathbf{J}_\omega \) and \( \mathbf{J}_e \) have to be estimated or approximated because of unknown feature depth [8]. Let them be represented by \( \hat{\mathbf{J}}_\omega \) and \( \hat{\mathbf{J}}_e \). In order to drive \( s \) to its desired value \( s^* \), we set \( v_r \) as constant and control \( \omega_r \) as

\[
\omega_r = -\hat{\mathbf{J}}_\omega^+ \left( \lambda (s - s^*) + \hat{\mathbf{J}}_e v_r \right),
\]

where \( \lambda \) is a positive gain, \( \hat{\mathbf{J}}_\omega^+ \) is the pseudo-inverse of \( \hat{\mathbf{J}}_\omega \) and \( (s - s^*) \) is the error [8]. Since we only control \( \omega_r \), only one feature derived from all line segments is sufficient.

The feature \( s \) is scalar, the average of the x-coordinates of the points of intersection of the \( m \) matched lines and their respective normal from the origin as in [4]. We represent the line on which the line segments lie in polar form as

\[
X \cos \theta + Y \sin \theta - \rho = 0
\]

as shown in Fig. 11b, and \( X = \rho \cos \theta \) gives the x-coordinate of the point. Hence,

\[
s = \bar{X} = \frac{1}{m} \sum_{i=1}^{m} \rho_i \cos \theta_i.
\]

The interaction matrix for line segments has been derived in [4] and it takes the form

\[
\hat{J}_{IL} \succeq 0, \quad \hat{J}_{oL} \succeq \frac{1}{m} \sum_{i=1}^{m} \left( \cos^2 \theta_i - \rho_i^2 \cos(2\theta_i) \right),
\]

where \( (\rho_i, \theta_i) \) are the line parameters (as shown in Fig. 11b) of the matched lines between \( I^c \) and \( I_{k+1}^R \), and \( \hat{J}_{IL} \) and \( \hat{J}_{oL} \) are the approximated Jacobians for translational velocity.

![Feature Selection and Control](Image)

**Fig. 9** Localization in the optimal path. The location of the current image \( I^c \) is between the reference images \( I_k^R \) and \( I_{k+1}^R \).
Creation of 2D Occupancy Grid Map

The depth camera provides an image of resolution 320 × 240 at 20 fps. The intrinsic parameters [22] of the depth camera are obtained from the depth camera calibration². We first take a horizontal central band of the depth image, 75 pixels high, as shown in Fig. 12b. This ensures that we can see most of the obstacles in the FOV of the robot while eliminating most of the clutter of the floor and ceiling plane shown in Fig. 12. Next, we convert the depth image into a local 2D grid map in the robot’s frame of reference. The 2D grid map is obtained by projecting the depth image onto the horizontal plane passing through the center of the RGB camera. In Fig. 12c, it is shown by blue points. Objects that are near to the robot may be missed because the working range of depth camera starts from 40cm. Therefore, we always remember obstacles that were previously observed until they pass behind the robot. In order to keep these obstacles in the local grid map, we use odometry to transform their positions (if within 2.5m of the robot) from the previous grid map to the current one. In Fig. 12c, this is shown by green points. The accuracy of the odometry in this range is sufficient for our purpose.

From the local 2D grid map, we created a low-resolution local occupancy grid map of size 4 × 80. The horizontal width of each grid cell is 0.1m. The first row corresponds to objects in the range 0 − 0.5m, the second row corresponds to range 0.5 − 1.5m, the third row to 1.5 − 2.5m and last row to 2.5 − 4m. In Fig. 12c, these rows are shown by white, cyan, yellow and purple bars respectively. We do not consider objects beyond 4m. The first row belongs to the region that is not in the current field of view of the depth camera. Objects in this region are remembered from the previous views. The choice of this grid configuration is sufficient for our indoor setting with corridors and free-space. Algorithm 2 presents the details of creating a 2D grid map from the depth image. Algorithm 3 describes creation of the occupancy grid from the 2D grid map (Fig. 14).

²http://wiki.ros.org/pepper/Tutorials/Calibration

and rotational velocity respectively. This IBVS system has global asymptotic stability as discussed in [3].

In order to smooth the rapid steering actions when switching between reference images, a feed-forward command is also added to \( \omega_r \). The feed-forward term is based on shared lines of \( I^c \) with \( I_{k+2}^R \). Let \( s_c, s_{k+1}^c \) and \( s_{k+2}^c \) be the computed feature (10) of \( I^c, I_{k+1}^R \) and \( I_{k+2}^R \) respectively. Then, the error term w.r.t \( I_{k+1}^R \) is \( s_c - s_{k+1}^c \) and w.r.t \( I_{k+2}^R \) is \( s_c - s_{k+2}^c \).

The final expression for the rotational velocity can be computed as

\[
\omega_r = -\frac{\lambda}{J_{ol}} \Big( h_1(s_c - s_{k+1}^c) + h_2(s_c - s_{k+2}^c) \Big), \tag{12}
\]

where \( h_1 \) and \( h_2 \) are positive weights such that \( h_1 + h_2 = 1 \). \( J_{ol} \) is calculated using the line parameters of the shared lines among \( I^c \), \( I_{k+1}^R \) and \( I_{k+2}^R \) in \( I^c \). Let \( n(I_k, I_y) \) be the number of lines matched between the images \( I_k \) and \( I_y \). Unlike using a fixed value as in [4], we calculate \( h_1 \) and \( h_2 \) automatically based on the number of lines matched with the current image as

\[
h_1 = \frac{n(I^c, I_{k+1}^R)}{n(I^c, I_{k+1}^R) + n(I^c, I_{k+2}^R)}, \quad h_2 = \frac{n(I^c, I_{k+2}^R)}{n(I^c, I_{k+1}^R) + n(I^c, I_{k+2}^R)}, \tag{13}
\]

which assigns higher weight dynamically to the reference image that is closer to \( I^c \) leading to smoother control around switching.

b) Obstacle / Collision Avoidance using Depth Sensor

We use the depth camera and a local 2D occupancy grid map, which we search for a collision-free path. IBVS controller navigates in local free-space.

The depth camera provides an image of a perspective camera (blue) with its optical axis perpendicular to the axis of robot rotation, and a representation of line in polar form [4].
Algorithm 2 2D grid map in the current robot frame of reference using depth image, previous grid and odometry.

INPUTS:
- \( f_x, f_y, u_x, u_y : f_x \) and \( f_y \) are the focal length and \((u_x, u_y)\) is the principal point of the depth camera in pixels (Intrinsic Parameters).
- \( t_x, t_y, t_z \): translation from depth to top RGB camera.
- \( d_x, d_y, d_0 \): 2D transformation from current frame to previous frame based on odometry.
- \( I^d \): current depth image. Pixels where depth cannot be computed have the value NaN.
- \( h \): scan height = 75 pixels.
- \( prev\_grid \): 2D grid map of previous frame,
- \( gd \): maximum depth considered in \( prev\_grid \) for transformation into current grid map = 2.5m.

OUTPUT:
- \( grid\_map \): 2D grid map in the current frame.

# grid\_map in the current frame from the current depth image.
Set all positions in grid\_map as FREE.

For each pixel \( i \) in row \( j \) DO
If \( I^d(j, i) \) is not NaN calculate
- \( x = \text{round} \left( \frac{x_p}{f_x} I^d(j, i) + t_x \right) \)
- \( z = \text{round} \left( \frac{z_p}{f_z} I^d(j, i) + t_z \right) \)
grid\_map(z, x) = OCCUPIED.

# transformation of obstacles from previous to current frame
For each occupied cell \((z_p, x_p)\) of grid\_map in the range of \( z_p \in (0 \rightarrow D_z) \) DO
- \( x = \text{round} \left( x_p \cos(d_0) - z_p \sin(d_0) + d_x \right) \)
- \( z = \text{round} \left( x_p \sin(d_0) + z_p \cos(d_0) + d_z \right) \)
If \( z > 0 \) grid\_map(z, x) = OCCUPIED.

Algorithm 3 Occupancy grid from 2D grid map.

INPUTS:
- \( grid\_map \): Current 2D Grid Map.
- \( D_x \): size of grid cell in X (in our case 10 cm).
- \( D_z = [D_z_1, D_z_2, D_z_3, D_z_4] = [1, 1.5, 2.5, 4] \): depth values (in Z direction) used to create low-resolution occupancy grid.

OUTPUT:
- \( occup\_grid \): Occupancy grid (in our case it is of size \( 4 \times 80 \)).

Set all elements in occup\_grid to FREE.
For each occupied cell \((z_p, x_p)\) of grid\_map in the range of \( z_p \in (0 \rightarrow D_z) \) DO
- \( x \): position of \( x_p \) in occupancy grid as integer number.
If \( z_p \in (0, D_z_1) \) occup\_grid(0, x) = OCCUPIED
If \( z_p \in (D_z_1, D_z_2) \) occup\_grid(1, x) = OCCUPIED
occup\_grid(2, x) = OCCUPIED *
occup\_grid(3, x) = OCCUPIED *
If \( z_p \in (D_z_2, D_z_3) \) occup\_grid(2, x) = OCCUPIED
occup\_grid(3, x) = OCCUPIED *
If \( z_p \in (D_z_3, D_z_4) \) occup\_grid(3, x) = OCCUPIED

* Note: If cell at \( x \) is occupied at a nearer row, set it occupied in all farther rows at \( x \). This is done to simplify the free-space detection at each row (Algorithm 4).
Algorithm 4 Computation of position of drivable-space.

**INPUTS:**
- *occup_grid*: Occupancy grid (in our case it is of size $4 \times 80$).
- *W_r*: width of the robot in the grid cell (in the unit of grid cells).

**OUTPUTS:**
- $f_i$: position of the center of the nearest drivable free-space in each row of *occup_grid*. In our case, it is $4 \times 1$ vector (Fig. 14c).
- $ds$: flag indicates the particular grid cell is drivable. In our case, it is $4 \times 1$ vector.

**FOR** each row $i$ in *occup_grid* from farthest to closest **DO**

# First search for drivable free-space around the center
- Set $ds(i) =$ TRUE
- Set center of row as origin
- Compute the number of FREE grid cells between the origin and the first OCCUPIED grid in both sides. Let them be $nL$ (for left side) and $nR$ (for right side) as shown in Fig. 13.

**IF** $(nL + nR) \geq W_r$ **THEN**
- Drivable free-space found around the center (Fig. 14a)
  - IF $\left(\frac{nL - Wr}{2} \text{ and } \frac{nL + Wr}{2} \right)$ $f_s(i) = 0$ **#** move straight
  - IF $\left(\frac{nL - Wr}{2} \text{ and } \frac{nL + Wr}{2} \right)$ $f_s(i) = \left(\frac{nL - Wr}{2} \text{ and } nR \right) \#$ turn left
  - IF $\left(\frac{nL - Wr}{2} \text{ and } \frac{W_r - nL}{2} \right)$ $f_s(i) = -\left(\frac{W_r - nL}{2} \right) \#$ turn right
  - ELSE $f_s(i) = nL - nR$ **#** turn according to $f_s$

**ELSE** # when $(nL + nR < W_r)$
- # Check for drivable free-space on either side (Fig. 14b-c) **and** select the nearest one from the origin if it exists.
- # Compute the number of OCCUPIED grid cells between the last FREE grid and next FREE grid cell in each side. Let them be $oL$ (for left side) and $oR$ (for right side) as shown in Fig. 13.

**IF** $(oR > thres \text{ and } oL > thres)$ **THEN** # large obstacle in front
- Set $ds(i) =$ FALSE

**ELSE** # Compute the number of FREE grid cells after the last OCCUPIED grid cell counted in each side. Let them be $nL'$ (for left side) and $nR'$ (for right side) as shown in Fig. 13.

**IF** $(nR' < W_r \text{ and } nL' < W_r)$ **THEN**
- Set $ds(i) =$ FALSE

**ELSE**
- Let $f_r = \left(\frac{W_r}{2} + oR\right)$ and $f_L = \left(\frac{W_r}{2} + oL\right)$
  - IF $(nR' \geq W_r)$ $f_s(i) = f_r$
  - IF $(nL' \geq W_r)$ $f_s(i) = f_L$
  - IF $i$ is the farthest grid or $ds(i + 1) =$ FALSE
    - IF $(oR < oL \text{ and } nL > W_r)$ $f_s(i) = f_r$
    - IF $(oR \geq oL \text{ and } nR > W_r)$ $f_s(i) = f_L$
  - **ELSE**
    - IF $(f_r \text{ and } f_s(i + 1) \text{ have same sign})$ $f_s(i) = f_L$
    - **ELSE** $f_s(i) = f_r$

Control Command for Free-space Navigation  For deriving the robot’s rotational velocity, we consider that a row in the occupancy grid cell is drivable if there are free adjacent grid cells in the horizontal direction whose width is at least the width of the base of the robot (Fig. 13). The computation of free-space is done by searching for the nearest drivable free-space in each row of the occupancy grid map from its center as described in Algorithm 4. The output of Algorithm 4 is the position of the center of drivable free-space for each row in the occupancy grid map and information about whether the particular grid cell is drivable. These centers are transformed back into the image space so as to compute the rotational velocity for each center based on IBVS. The calculation of the rotational velocity is explained below.

From [8] and (7), the interaction matrix ($L_\chi$) of the normalized image plane point $(x, y)$ that relates its velocity to robot velocity $(v_r, \omega_r)$ is

$$L_\chi = \left[ \frac{x}{Z} - \left(\frac{1}{2} + 1 + x^2\right) \right] = \begin{bmatrix} J_{vp} & J_{\omega p} \end{bmatrix},$$

where $Z$ is the depth of the point and $J_{vp}$ and $J_{\omega p}$ are the Jacobians for translational velocity and rotational velocity respectively. We can approximate Jacobians for the case $\delta \ll Z$ as follows

$$\hat{J}_{vp} = \frac{x}{Z} \text{ and } \hat{J}_{\omega p} = 1 + x^2.$$  (15)

In order to drive robot to the center of free-space, from (9) and (15), we obtain

$$\omega_r = -\frac{\lambda(x - 0) + \frac{s}{2} v_r}{1 + x^2},$$

where $s = x$ and $s^* = 0$. The final rotational velocity is computed by a weighted average of the velocities corresponding to the two farthermost drivable grid cells as described in Algorithm 5. We have used selective weighting for the smooth motion of the robot that was determined experimentally. This configuration is sufficient to avoid collision with lateral and frontal obstacles in the path. The weights chosen for our experiments are presented in Algorithm 5.

c) Fusion of Control  During navigation, the robot has two control commands: from the line-segment based navigation module, and from the free-space navigation module. The Pepper robot cannot turn its head while moving, a limitation in its internal controller, the head will always face straight ahead. Hence, we cannot fuse both control commands as in [9], where the pan angle of the camera was also controlled during obstacle avoidance. If $\omega_{rL}$ is the rotational velocity obtained from line-segment navigation as in (12), $v^*_r$ and $\omega^*_r$ are the forward and the rotational velocity obtained from free-space navigation as discussed in Algorithm 5, and $d$ is the mean depth of the 2D grid cells discussed in Algorithm
Fig. 13 Robot occupancy grid. Drivable-space in the row of occupancy grid a when there is free-space in front of the robot and b when there is an obstacle in front of the robot. The white blob represents the robot’s body. The symbol ⊗ in b represents the case where the free-space is not large enough for the robot to move forward. c Occupancy grid map with drivable-space. (Ref. Algorithm 4). The red dot represents the origin. The colored bars in c are the rows of the occupancy grid map. The colored dots in each row represents the nearest free space from the origin in each row through which the robot can pass safely. \( f_s(x) \) is position of the center of the nearest drivable free-space in row \( x \). These positions in each row are used to compute rotational velocity in Algorithm 5.

\[
v_r = \begin{cases} 
0 & \text{if } v_r^* = 0 \\
v_r & \text{elsewhere}
\end{cases}
\]
\[
\omega_r = \begin{cases} 
0 & \text{if } v_r^* = 0 \\
H \omega_r^* + (1 - H) \omega_r & \text{elsewhere}
\end{cases}
\]  
where \( H \) is defined as
\[
H = \begin{cases} 
0.5 & \text{if } |\omega_r^*| < 0.01 \text{ or } |\omega_L^*| > 0.15 \\
1 & \text{elsewhere}
\end{cases}
\]

The velocity commands from (17) ensure that the robot moves towards the destination with enough clearance from the obstacles to avoid collision. \( \omega_r^* \) in (18) is calculated from common matched lines among the reference images \( I_k^R, I_{k+1}^R, \text{ and } I_{k+2}^R \). From \( \omega_r^* \), we can know whether the navigation path between the current reference images is straight or not. \( H \) is set to zero when the free-space navigation tells the robot to continue straight or when the robot is commanded to perform turning by the line-based navigation. We have to stop free-space navigation while turning because of the limited FOV of the depth camera, which cannot see the space in advance before turning. In such a case, the robot will stop if there is a potential collision detected by the Pepper robot’s own internal safety module.

3 Experimental Results

The image resolution in the experiments was 320 × 240 pixels. Throughout the experiment, the head pitch and roll angles of the Pepper were kept constant (same as that used in the mapping phase). Image acquisition and motion commands used the NaoQi API [32].

First, we performed experiments to validate the free-space navigation using the depth sensor, and image-based navigation with collision avoidance. Finally, a long-duration experiment with the entire topological navigation system was performed in an open-plan office environment. We have also included a supplementary video with this manuscript, which demonstrates the qualitative measure of the robustness of our method.

3.1 Navigation in Free-space with only Collision Avoidance

In this experiment, the Pepper robot is allowed to move freely in the corridor finding drivable-space as discussed.
Algorithm 5 Computation of rotational velocity based on drivable free-space.

INPUTS:
- \( f_s \): center of the drivable free-space, which is a 4x1 vector corresponding to each row in the occupancy grid map (white dots in Fig. 13c).
- \( ds \): flag indicates the particular grid cell is drivable. In our case, it is \( 4 \times 1 \) vector.
- \( D_t = [D_{z1}, D_{z2}, D_{z3}, D_{z4}] = [0.25, 1, 2, 3.25] \): nominated depth value associated with the rows of occupancy grid map.
- \( D_x \): resolution of grid in x-direction (in our case 0.1m).
- \( v_r \): forward velocity of the robot.

OUTPUTS:
- \( \omega^*_r \): rotational velocity required to move in free-space avoiding collision.
- \( v_{rd} \): forward velocity of robot.
- \( d \): mean depth of the occupancy grid. It is used to calculate final \( \omega_r \).

FOR \( i = 0 \rightarrow 3 \) from nearest to farthest grid row DO

\[
x = f_s[i] D_t[i] / D_x \text{ # visual feature in image space}
\]

\[
\omega[i] = -\frac{(x - o) + v_{rd}}{1 + x^2} \text{ # From (16)}.
\]

\[
\omega^*_r = \begin{cases} 
0 & \text{if } ds(1) = \text{FALSE or } ds(0) = \text{FALSE} \\
0.4\omega[0] + 0.6\omega[1] & \text{if } ds(2) = \text{FALSE} \\
0.5\omega[1] + 0.5\omega[2] & \text{if } ds(3) = \text{FALSE} \\
0.6\omega[2] + 0.4\omega[3] & \text{elsewhere}. 
\end{cases}
\]

\[
v_{rd} = \begin{cases} 
0 & \text{if } ds(1) = \text{FALSE or } ds(0) = \text{FALSE} \\
v_r & \text{elsewhere}.
\end{cases}
\]

\[
d = \begin{cases} 
0.6D_{z2} + 0.4D_{z1} & \text{if } ds(2) = \text{FALSE} \\
0.5D_{z3} + 0.5D_{z2} & \text{if } ds(3) = \text{FALSE} \\
0.6D_{z4} + 0.4D_{z3} & \text{elsewhere}. 
\end{cases}
\]

Note: if \( ds(1) = \text{FALSE or } ds(0) = \text{FALSE} \), we have to stop the robot because the obstacle will be close to the robot. \( d \) is used in the fusion of control as discussed in (17) - (18).

in Section 2.2.4 with the base forward velocity of 0.18m/s. In the first experiment as shown in Fig. 15, the robot automatically found the drivable free-space avoiding collision with people and the door and successfully entered into the room and stopped when it could not move further.

In the second experiment as shown in Fig. 16, the robot is moving straight along a corridor of length 35m avoiding collision with the walls and furniture, and stopped after entering into the room at the end. Some snapshots from the navigation are presented in Figs. 15a and 16a. In Figs. 15b and 16b, the peaks in the graph shows that the robot is turning in the direction of drivable free-space.

3.2 Image-based Navigation with Collision Avoidance

This navigation process is divided into teaching the path and online navigation. The teaching step involves showing the Pepper robot around the navigation path by holding the robot’s hand as shown in Fig. 4. During this process, the robot stores the images as it moves. The selection of the reference images is done offline from this image sequence as discussed in Section 2.1.1. The navigation path consists of 119 reference images selected from the sequence of 1700 images representing the two laps of the corridor path of 42m as shown in Fig. 17. During navigation, the forward velocity was set to 0.18m/s and reduced to 0.06m/s when turning, or stopped when a collision-free path is not available as given by (17). Rotational velocity was controlled by the navigation algorithm (17). The turnings are automatically detected by observing the commanded rotational velocity. The threshold we have chosen for this is 0.1rad/s. Figure 18 shows the localization in the image memory of the optimal path at the start and the end of the navigation. Figure 19a shows images acquired during navigation. Figure 19b shows the commanded rotational velocity. The plot in green is the output from the free-space navigation module, whereas the plot in black is the fused rotational velocity command.

3.3 Navigation in the Topological Graph

Navigation in the topological graph is the extension of the navigation performed above. First, we created a topological graph. The initial step in the navigation is path-planning, which outputs a sequence of reference images from source to destination of the optimal path. Once we have these reference images, the navigation process from source to destination is the same as in Section 3.2. In the experiments, the base linear velocity was set to 0.18m/s and reduced to 0.06m/s when turning or stopped when a collision-free path is not available as given by (17). We have also performed successful navigation for different base linear velocities ranging from 0.05m/s to 0.25m/s.

In this work, we are interested in goal-oriented navigation, i.e. robot successfully reaching the goal. The most appropriate measure for a robot that performs a task is how well it performs the task. The task or mission we set for
the robot is obstacle-free navigation between waypoints. Therefore, we have used mission success over many missions as an end-to-end performance of our embedded visual navigation system.

### 3.3.1 Creation of Topological Graph

First, we drive the robot around by holding its hand and capturing images. The navigation environment is an open-plan office environment with many internal glass windows and doors. We captured images in the mid-afternoon with most of the lights turned on. The area has many external windows that let in direct sunlight. The reference images are selected from the acquired sequence automatically as discussed in Section 2.1.1. Altogether, our navigation path consists of 451 reference images distributed over the 150m path. The concentration of reference images is highest around the turnings and regions where the illumination varies quickly. Some of the reference images are shown in Fig. 20. We consider 8 top-level nodes in the environment, connected by 11 bidirectional edges as shown in Fig. 21. Each edge is associated with two sets of reference images for traveling in opposite directions. A semi-manual approach is used to allocate reference images to edges. Thus, reference image collection is manual, reference image selection is automatic, and creating the topological data-structure is semi-automatic.

### 3.3.2 Navigation in the Graph

Once we have the topological graph, we can perform navigation. The first step is global localization in the topological graph. In Fig. 22a, the robot is localized in between the nodes $N_b$ and $N_f$ in the edge $E_{bf}$ shown in cyan. The robot is facing towards node $N_f$. The destination node we have selected is $N_D$ that is shown in blue. Once we have an initial location and destination, we performed A* search to obtain the optimal path for
navigation. The optimal path is shown in red in Fig. 22b. The start node \((N_S)\) is shown in green and the destination node \((N_D)\) is shown in blue. The optimal path consists of nodes \(N_S, N_I\) and \(N_D\) in sequential order. The reference images of the edges \(E_{SI}\) and \(E_{ID}\) (shown in red) are sequentially extracted from the image memory. Once we have the set of reference images, we performed navigation as discussed in Section 3.2. Some images for navigation are shown in Fig. 23.

Figure 24 shows an instance of navigation where the robot has to initially turn \(180^\circ\) to navigate the optimal path. The robot’s initial location is in the cyan edge \(E_{bf}\) with the robot facing from node \(N_b\) to \(N_f\), whereas the destination node \((N_D)\) is shown in blue (Fig. 24a). The chosen optimal path consists of edges \(E_{SI}\) and \(E_{ID}\) in sequential order with start node \(N_S\) and destination node \(N_D\) (Fig. 24b). The direction of \(E_{bf}\) and \(E_{SI}\) is just opposite (Fig. 24). Therefore, the robot turned \(180^\circ\) at the beginning of navigation in order to face towards the optimal path shown by red arrows in Fig. 24b.

Some images during navigation in the environment are shown in Fig. 25. We performed a long-term test of continuous navigation while avoiding collisions. Once the robot reached the destination, we randomly selected a new one. This navigation was conducted over 15 days visiting more than 50 destination nodes and traveling more than \(1200m\) with a success rate over 80%. We consider the navigation as a success if the robot successfully reaches the final destination. The navigation was performed in a busy open-plan environment with people passing by, displacement of the objects and furniture, changing illumination, shadows due to sunlight etc. Our navigation framework is robust in the presence of occlusion, blurring, and illumination changes.

The most common failure mode was the robot becoming lost. This occurred when the robot was unable to find the
Fig. 17 Some reference images from the navigation path

correct reference images due to large occlusions, or drastic changes in the illumination compared to mapping time due to switching off lights or changes in the ambient light from outside. These conditions result in poor performance of the line matching, especially in low textured places. Furthermore, the depth image also becomes noisy in the presence of bright sunlight causing the poor performance of the free-space navigation. Despite these limitations, our approach is still able to perform useful navigation in the indoor environment.

Our proposed navigation system is able to run on the onboard CPU. The global localization in the topological graph (Section 2.2.1) took around 14.5s, which involves matching the line segments of the current image with that of 451 reference images in the graph. The image-based navigation took 59ms (89ms when there is reference image switching) on average. The mean time taken by free-space navigation was 35ms. The navigation loop (from acquisition to generate final command) took 112ms (153ms when there is reference image switching) on average.

Our goal is to develop a navigation system that can run robustly real-time onboard in the Pepper robot, which has limited onboard computing power and average quality sensors for navigation. These limitations of our platform

Fig. 18 Localization in the image memory of the optimal path at the start and the end of navigation
Fig. 19  a Some images acquired during navigation and b commanded rotational velocity
limit meaningful comparisons against available solutions in the literature. We are not able to run monocular ORB SLAM [28] in real-time onboard. Furthermore, the tracking of points fails too soon often due to varying illumination level and motion blur, which breaks standard Visual SLAM algorithms using the point features. However, the line segment detection and matching algorithm that we have used is robust enough to handle this situation to run in
real-time. The work of [33] requires a 2D metric map and external computational resources. The recent deep learning-based teach-and-repeat navigation such as [7] is not suitable to run onboard in the resource-constrained platform like ours.

4 Discussions and Limitations

The presented results show the viability of our approach. The free-space navigation complemented the image-based navigation so that the robot moves with sufficient clearance.

Fig. 21  Topological Map of our environment

Fig. 22  a Start edge (in cyan) and destination node (in blue). The robot is facing in the direction of the node in cyan. b Retrieval of the reference images in the optimal path (red) from source (green) to destination (blue)
from walls and furniture. This is necessary for safe and continuous smooth motion. Even with the limited capabilities of the sensors on the Pepper robot, we are able to demonstrate navigation based on a teach-and-repeat paradigm. Our core navigation module can run real-time on a robot with quite limited processing power and memory. In all the experiments, we have set the posture of the robot at the beginning and fixed head pitch and roll angles to 0. During navigation, only the base velocity has been controlled. The balancing of the body and limbs during navigation is done by the robot’s internal software, which sometimes causes the robot to lean its body. Our method is able to cope with this leaning.

The mapping process is intuitive and straightforward: holding the hand of Pepper robot and leading the robot around a path as shown in Fig. 4, then creating image memory offline. The image memory required for the navigation consists of just a few selected reference images of the path. In fact, the detected line segments with their descriptors are sufficient. Hence, image memory is very compact in our case. Once this image memory has been created, we organize it as a topological graph. Once we
Some images during the navigation in the topological graph

have a topological map, we can perform navigation using the RGB and depth cameras. The line-based navigation can be easily interchanged with point features, combinations of both, or other similar image-based navigation methods such as [3, 5, 30].

Our approach for topological navigation can be easily implemented in the indoor environment. The topological graph is simple and flexible. It can be easily extended because adding new nodes and edges only require reference images of the new path and its link with the current graph.
We can also easily change the reference images of particular edges to incorporate significant changes in the appearance of the specific locations in the environment.

Our experiments have shown that reliable navigation can be performed without a global 3D map. The integration of image-based navigation with free-space navigation ensures that the robot avoids collisions and moves with appropriate clearance from obstacles such as walls and furniture. The robot is able to move along a narrow corridor without becoming stuck, which was a frequent problem using only [4].

However, our approach has some limitations. The free-space navigation performs poorly in the presence of bright sunlight due to increased noise in the depth image. The performance of line segment matching algorithm is the limiting factor for image-based navigation. The line matching algorithm that we have used is quite robust for typical indoor environments that have sufficient texture. It is robust enough to handle motion blur, moderate occlusions and some change in the illumination. The line matching algorithm’s poor performance is due to the drastic change in lighting from the mapping time, low texture in the scene, or very large occlusions that block most of the scene. Most of the failure cases in our experiments come from this poor performance of the line matching algorithm or noisy depth image due to bright sunlight in the path. Using multiple features as in [3, 30] can increase the robustness of the image-based navigation. However, the computing power of our platform limits our capacity to use these methods. Furthermore, the Bag-of-Words approach using line segments similar to [18] can be easily incorporated for the global localization to increase the robustness to aliasing if required.

Another limitation is that we stop free-space navigation when the Pepper robot is making a sharp turn so as to maintain the desired FOV of the RGB camera due to our requirement to keep the head fixed while the robot is moving. In this case, we rely on the robot’s internal safety module that uses laser and sonar sensors to stop it in case of possible collisions. Nevertheless, our free-space navigation is modular and can be easily integrated with methods similar to [3, 30]. During navigation, the robot stops if it could not find drivable free-space. The other unexpected and dangerous collisions are handled by the robot’s internal safety module that uses lasers, sonars, and touch. This internal safety module will stop the robot and overrides any other motion commands until the robot is out of danger.

Since Pepper is a social robot, it can interact with people. In fact, it is designed to do so. The navigation can take advantage of this social context. Whenever the robot is unable to recover from being lost, it could ask people for help to guide it towards its next goal as shown in Fig. 4.

Once the robot re-localizes in the graph when it is being rescued, it can continue navigation on its own. In future, adding this capability increases the robustness of navigation in social scenarios.

5 Conclusions

We have presented a complete framework for indoor topological navigation with collision avoidance for a robot like Pepper. We have demonstrated that robust and useful navigation can be performed using the proposed framework without 3D maps or 3D pose estimation. Our core navigation module is computationally light-weight enough to run onboard. The method is general enough to be used for any non-holonomic wheeled robot with RGB and depth cameras.

In this work, we have considered the Pepper robot as a non-holonomic robot; however, it is capable of moving in a lateral direction. Incorporating this motion capability and using probabilistic filters to estimate the final control velocity from the obtained rotational velocity to make the motion smoother will enable future improvements. Such filtering will help to overcome uncertainties in the navigation caused by the poor performance of feature matching or tracking algorithms, such as due to low scene texture. In future, Pepper, being a social robot, could ask for human support to drive it to a known location when it is lost, continuing the navigation as soon as it re-localizes in the image memory. This can be a future challenge to improve the navigation robustness of robots like Pepper in busy and complex social contexts. Furthermore, exploring the robust vision that can handle different lighting conditions will be future work.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10846-021-01390-6.

Authors Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Suman Raj Bista. The manuscript was written by Suman Raj Bista and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding This research was supported by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016) and funded by the Queensland Government under an Advance Queensland Grant.

Code Availability The complete software is open-sourced. The source code is available on https://github.com/qcr/pepper_navigation.git

Declarations

Conflict of Interests Distinguished Professor Peter Corke (One of the authors of the manuscript) is in the Board of Governors of this journal.
References

1. Akinlar, C., Topal, C.: EDLines: A real-time line segment detector with a false detection control. In: Pattern Recognition Letters 32.13, pp. 1633–1642. ISSN: 0167-8655. (2011). https://doi.org/10.1016/j.patrec.2011.06.001

2. de Lima, D.A., Victorino, A.C.: A Hybrid Controller for Vision-Based Navigation of Autonomous Vehicles in Urban Environments. In: IEEE Transactions on Intelligent Transportation Systems 17.8, pp. 2310–2323. ISSN: 1524-9050 (2016). https://doi.org/10.1109/TITS.2016.2519329

3. Bista, S.R., Giordano, P.R., Chaumette, F.: Combining line segments and points for appearance-based indoor navigation by image based visual servoing. In: IEEE/RSJ international conference on intelligent robots and systems. IEEE, pp. 2960–2967 (2017). https://doi.org/10.1109/IROS.2017.8206131

4. Bista, S.R., Giordano, P.R., Chaumette, F.: Appearance-based indoor navigation by IBVS using line segments. In: IEEE robotics and automation letters 1.1, pp. 423–430. ISSN: 2377-3766 (2016). https://doi.org/10.1101/LRA.2016.2521907

5. Bista, S.R., Giordano, P.R., Chaumette, F.: Appearance-based indoor navigation by IBVS using mutual information. In: IEEE international conference on control, automation, robotics and vision. pp. 1–6 (2016). https://doi.org/10.1109/ICARCV.2016.7838760

6. Bonin-Font, F., Ortiz, A., Oliver, G.: Visual navigation for mobile robots: A survey. In: Journal of intelligent and robotic systems 53.3, pp. 263–296. ISSN: 1573-0409 (2008). https://doi.org/10.1007/s10846-008-9235-4

7. Camara, L.G., Pivonka, T., Ijke, M., Gábert, C., Kosnar, K., Preucil, L.: Accurate and robust teach and repeat navigation by visual place recognition: A CNN approach. In: IEEE/RSJ International conference on intelligent robots and systems (2020)

8. Chaumette, F., Hutchinson, S.: Visual servo control, Part I: Basic approaches. In: IEEE robotics and automation magazine 13.4, pp. 82–90. ISSN: 1070-9932 (2006). https://doi.org/10.1109/MA.2006.250573

9. Cherubini, A., Chaumette, F.: Visual navigation of a mobile robot with laser-based collision avoidance. In: The international journal of robotics research 32.2, pp. 189–205 (2013). https://doi.org/10.1177/0278786912460413

10. Clement, L., Kelly, J., Barfoot, T.D.: Robust monocular visual teach and repeat aided by local ground planarity and color-constant imagery. In: Journal of field robotics 34.1, pp. 74–97. ISSN: 1556-4959 (2017). https://doi.org/10.1002/rob.21655

11. Dame, A., Marchesh, E.: Using mutual information for appearance-based visual path following. In: Robotics and autonomous systems 61.3, pp. 259–270. ISSN: 0921-8993 (2019). https://doi.org/10.1016/j.robot.2012.11.004

12. Davison, A.J., Reid, I.D., Molton, N.D., Stasse, O.: MonoSLAM: Real-time single camera SLAM. In: IEEE Transactions on pattern analysis and machine intelligence 29.6, pp. 1052–1067. ISSN: 0162-8828 (2007). https://doi.org/10.1109/TPAMI.2007.1049

13. Delfin, J., Becerra, H.M., Arefchaeva, G.: Humanoid navigation using a visual memory with obstacle avoidance. In: Robotics and autonomous systems 109, pp. 109–124. ISSN: 0921-8890 (2018). https://doi.org/10.1016/j.robot.2018.08.010

14. Diosi, A., Segvic, S., Remazeilles, A., Chaumette, F.: Experimental evaluation of autonomous driving based on visual memory and image based visual servoing. In: IEEE Transactions on intelligent transportation systems 12.3, pp. 870–883. ISSN: 1524-9050 (2011). https://doi.org/10.1109/TITS.2011.2122334

15. Folió, D., Cadena, V.: A redundancy-based scheme to perform safe vision-based tasks amidst obstacles. In: IEEE International conference on robotics and biomimetics. pp. 13–18 (2006). https://doi.org/10.1109/ROBIO.2006.340252

16. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. In: IEEE robotics automation magazine 4.1, pp. 23–33. ISSN: 1070-9932 (1997). https://doi.org/10.1109/100.580977

17. Furgale, P., Barfoot, T.D.: Visual teach and repeat for long-range rover autonomy. In: Journal of field robotics 27.5, pp. 534–560 (2010). https://doi.org/10.1002/rob.20342

18. Gálvez-López, D., Tardó, J.D.: Bags of binary words for fast place recognition in image sequences. In: IEEE Transactions on robotics 28.5, pp. 1188–1197. ISSN: 1552-3098 (2012). https://doi.org/10.1109/TRO.2012.2197158

19. Garcia-Fidalgo, E., Ortiz, A.: Vision-based topological mapping and localization methods: A survey. In: Robotics and autonomous systems 64, pp. 1–20. ISSN: 0921-8890 (2015). https://doi.org/10.1016/j.robot.2014.11.009

20. Gee, A.P., Mayol-Cuevas, W.: Real-time model-based SLAM using line segments. In: Proceedings of the second international conference on advances in visual computing. Springer-Verlag, pp. 354–363 (2006). https://doi.org/10.1007/11919629_37

21. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. In: IEEE Transactions on systems science and cybernetics 4.2, pp. 100–107. ISSN: 0536-1567 (1968). https://doi.org/10.1109/TSSC.1968.300136

22. Hartley, R., Zisserman, A.: Multiple view geometry in computer vision. Second. Cambridge University Press, Cambridge (2004). ISBN: 0521540518

23. von Hundelshausen, F., Himmelsbach, M., Hecker, F., Mueller, A., Wuerhse, H.-J.: Driving with tentacles: Integral structures for sensing and motion. In: Journal of field robotics 25.9, pp. 640–673. ISSN:1556-4959 (2008). https://doi.org/10.1002/rob.v25.9

24. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. In: IEEE International conference on robotics and automation. Vol. 1., pp. 496–501 (1985). https://doi.org/10.1109/ROBOT.1985.1087247

25. Labrosse, F.: Short and long-range visual navigation using warped panoramic images. In: Robotic and autonomous systems 55.9, pp. 675–684. ISSN:0921-8890 (2007). https://doi.org/10.1016/j.robot.2007.05.004

26. Micusik, B., Wildenauer, H.: Descriptor free visual indoor localization with line segments. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3165–3173 (2015). https://doi.org/10.1109/CVPR.2015.7298936

27. Mujahed, M., Fischer, D., Mertsching, B.: Admissible gap navigation: A new collision avoidance approach. In: Robotics and autonomous systems 103, pp. 93–110. ISSN: 0921-8890 (2018). https://doi.org/10.1016/j.robot.2018.02.008

28. Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: ORB-SLAM: A versatile and accurate monocular SLAM system. In: IEEE Transactions on robotics 31.5, pp. 1147–1163. ISSN: 1941-0468 (2015). https://doi.org/10.1109/TRO.2015.2463671

29. Seder, M., Petrovic, I.: Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In: IEEE International conference on robotics and automation. pp. 1986–1991 (2007). https://doi.org/10.1109/ROBOT.2007.363613

30. Segvic, S., Remazeilles, A., Diosi, A., Chaumette, F.: A mapping and localization framework for scalable appearance-based navigation. In: Computer vision and image understanding 113, pp. 172–187. ISSN: 1077-3142 (2009). https://doi.org/10.1016/j cviu.2008.08.005

31. Smith, F., Reid, I., Davison, A.: Real-time monocular SLAM with straight lines. In: Proc. British Machine Vision Conference. pp. 17–26 (2006)
32. SoftBank Robotics: Pepper-Documentation. Accessed: 12-02-19. http://doc.aldebaran.com/2-5/home_pepper.html (2016)
33. Suddrey, G., Jacobson, A., Ward, B.: Enabling a pepper robot to provide automated and interactive tours of a robotics laboratory. In: Australasian conference on robotics and automation (2018). arXiv:1804.03288
34. Swedish, T., Raskar, R.: Deep visual teach and repeat on path networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 1533–1542 (2018)
35. Von Gioi, R.G., Jakubowicz, J., Morel, J.-M., Randall, G.: LSD: A fast line segment detector with a false detection control. In: IEEE transactions on pattern analysis and machine intelligence 32.4, pp. 722–732 (2008). https://doi.org/10.1109/TPAMI.2008.300
36. Warren, M., Greeff, M., Patel, B., Collier, J., Schoellig, A.P., Barfoot, T.D.: There’s no place like home: Visual teach and repeat for emergency return of Multirotor UAVs during GPS failure. In: IEEE Robotics and automation letters 4.1, pp. 161–168. ISSN: 2377-3766 (2019). https://doi.org/10.1109/LRA.2018.2883408
37. Zhang, L., Koch, R.: An efficient and robust line segment matching approach based on LBD descriptor and pairwise geometric consistency. In: Journal of visual communication and image representation 24.7, pp. 794–805. ISSN:1047-3203 (2013). https://doi.org/10.1016/j.jvcir.2013.05.006
38. Zhang, L., Koch, R.einhard.: Hand-held monocular SLAM based on line segments. In: International machine vision and image processing conference 0, pp. 7–14 (2011). https://doi.org/10.1109/IMVIP.2011.011

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Suman Raj Bista is a Research Fellow at Australian Centre for Robotic Vision based at the Queensland University of Technology. He completed his PhD in 2016 with the Lagadic Group, INRIA Rennes Bretagne Atlantique/University of Rennes 1, France. He received his master’s degree in computer Vision from the University of Burgundy, France (2013) and his undergraduate degree in Electronics & Communication Engineering from Pulchowk Campus, Institute of Engineering, Tribhuvan University, Nepal (2009). He was also an Erasmus Visiting student in the Computer Science Department at the University of York, UK (2009). His research interests include Visual Navigation, Visual learning, Robotics Vision, and Optimisation.

Belinda Ward is a project manager at the Queensland University of Technology, leading a project in social robotics using Softbank’s Pepper robot platform. The aim is to understand the role of social robots in influencing human behaviour, and how enhanced robotic vision capabilities can help improve their interaction with people. Previously at CSIRO’s Autonomous Systems Laboratory, Belinda led projects to install a multi-user telepresence robot in the National Museum of Australia, and to provide science education using immersive 3D environments and augmented reality. Prior to that, Belinda worked in research strategy at Telstra’s Chief Technology Office and in technology innovation at Telstra Research Laboratories. She has undergraduate degrees in Mathematical Science and in Electrical and Electronic Engineering from The University of Adelaide, a master’s degree in Cognitive Science, and a Master of Commercial Law from The University of Melbourne.

Peter Corke is the distinguished professor of robotic vision at Queensland University of Technology, director of the QUT Centre for Robotics and former director of the ARC Centre of Excellence for Robotic Vision. He is the Chief Scientist of Dorabot and on the advisory boards for Emesent and LYRO Robotics. His research is concerned with enabling robots to see, and the application of robots to mining, agriculture, and environmental monitoring. He is well known for his robotics toolbox software for MATLAB and Python, the best selling textbook “Robotics, Vision, and Control”, MOOCs and the online Robot Academy. He is a fellow of the IEEE, the Australian Academy of Technology and Engineering, and the Australian Academy of Science. He has held a number of editorial board positions and held various visiting positions. He received his undergraduate and master’s degrees in electrical engineering and PhD from the University of Melbourne.