Real-Time Egocentric Navigation Using 3D Sensing

Justin S. Smith, Shiyu Feng, Fanzhe Lyu, and Patricio A. Vela

Abstract This chapter proposes a hierarchical navigation system combining the benefits of perception space local planning and allocentric global planning. Perception space permits computationally efficient 3D collision checking, enabling safe navigation in environments that do not meet the conditions assumed by traditional navigation systems based on planar laser scans. Contributions include approaches for scoring and collision checking trajectories in perception space. Benchmarking results show the advantages of perception space collision checking over popular alternatives in the context of real time local planning. Simulated experiments with multiple robotic platforms in several environments demonstrate the importance of 3D collision checking and the utility of a mixed representation hierarchical navigation system.

Justin S. Smith
Georgia Tech, School of Electrical and Computer Engineering, North Ave NW, Atlanta, GA 30332, e-mail: jssmith@gatech.edu

Shiyu Feng
Georgia Tech, School of Mechanical Engineering, North Ave NW, Atlanta, GA 30332 e-mail: shiyufeng@gatech.edu

Fanzhe Lyu
Georgia Tech, School of Electrical and Computer Engineering, North Ave NW, Atlanta, GA 30332 e-mail: fanzhe@gatech.edu

Patricio Vela
Georgia Tech, School of Electrical and Computer Engineering, North Ave NW, Atlanta, GA 30332 e-mail: pvela@gatech.edu

This work supported in part by NSF Awards #1400256 and #1849333.

1 Introduction

Navigation is an essential computational component of autonomous mobile robots, ensuring that the robot gets from one point in space to another. When deployed, it is commonly implemented as a two time-scale solution involving planning with real-time constraints on decision-making. The longer (or slower) time-scale process aims to find a global path from the current robot location to the desired terminal location and relies on a valid or sufficiently complete map of the world to navigate. The shorter (or faster) time-scale process maneuvers the robot within the world avoiding obstacles while striving to follow the global path. It is common for the global path to be approximately realizable due to uncertainty in the global map. Uncertainty arises from missing map data or modified world structure due to moved, introduced, or removed objects. In dynamic environments, the maneuvering should avoid other moving objects, which would typically be people, animals, cars, or robots, depending on the application. The faster process detects and adjusts the trajectory in response to deviations between locally sensed world geometry and presumed world map geometry, especially when the global trajectory violates the collision-free trajectory constraints based on the local sensor data. Given the domain and source of the map data used, the slower and faster planning processes are often called the *global* and *local* planners. A navigation system modulates and coordinates the information flow between the two planning processes. Figure 1 depicts a sample scenario with the robot starting on the left (black dot) and tasked to move to the right (red dot) The global path planner generates a candidate trajectory at the onset (solid green) based on available map information. In this case, only the walls are known. The local planner strives to match the global path, but must detour as unknown obstacles are identified. Depicted are two paths, one of which is realizable by a thinner robot (solid blue), and the other which is realizable by a wider robot (solid yellow). The wider robot would not be aware of the infeasibility of the plan until arriving at the impasse. A global re-plan would then provide a new, potentially feasible path based on the additional map information gained during navigation. This new path would be followed and modified as potential collision points are identified and avoided.

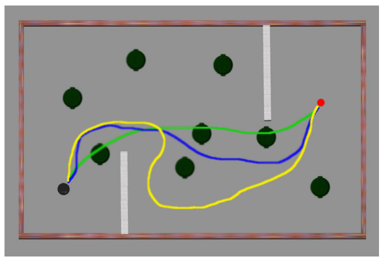


Fig. 1 Depiction of the complementary roles of the global planner (green trajectory) and local planners (blue and yellow trajectories) in the context of navigation.

Important considerations associated with a navigation system include the rate at which the global path is recomputed based on locally sensed information or state triggers, what information is shared between the two processes, what world representation is chosen, and what planning strategies are employed for the global and local planners. This chapter summarizes the research and findings related to the global planner and techniques for the local planner. For the latter, the discussion focuses on developments related to the contemporary availability of dense visual measurements of the 3D world structure as compared to the predominantly 2D laser scan measurements used by classical planning algorithms and employed in many local planning methods. The use of image-based measurements that provide depth or range information increases the cardinality of the data to process by two orders of magnitude or more. Classical laser scanning methods do not translate to or do not scale well with the increased sensory data when going from planar to spatial measurements (2D to 3D). This chapter describes alternative data representation, trajectory scoring, and collision detection schemes that improve on the weaknesses of classical methods while striving to be as compatible as possible with them. In doing so, we anticipate that many classical and modern navigation methods can be modified to work with modern dense imaging systems that provide depth or range information.

The chapter is organized to first cover global planning strategies (§2), followed by local planning methods (§3). The review of local planning methods discusses implementation modifications due to advances in robot sensing from laser scanners to dense 3D range-based sensors. Purely monocular, color camera sensing is excluded due to the loss of depth information and the inability of these methods to guarantee collision-avoidance when traveling within the field of view. Findings from neuroscience covered in §4 motivate a mixed method solution, whereby the global world representation and the local world representations differ. In particular, the local representation minimally processes the sensory data and operates in Marr's mid-level visual representation. The result is a perception space local planner whose design and integration with a global planner is described in §5 and contrasted to existing strategies described in the literature. Prior to implementing and evaluating the PiPS approach, §6 describes a navigation benchmark for evaluating mobile robot navigation algorithms. The experimental outcomes in §7 cover range-based navigation strategies using two mobile robot platforms and multiple environments. Experimental Monte-Carlo runs quantify the success of these methods relative to traditional approaches. Finally, Section §8 summarizes the chapter contents and provides concluding remarks.

2 Global Planning

Planning a robot trajectory, or synthesizing a control signal whose application leads to a robot trajectory, requires a representation of the world within the computer proper [1, 2, 3]. Initially, representation strategies may be split into those that attempt to preserve the underlying continuity or those that discretize the world for easier processing. The latter approach is most often taken on account of simplifications afforded by discretization.

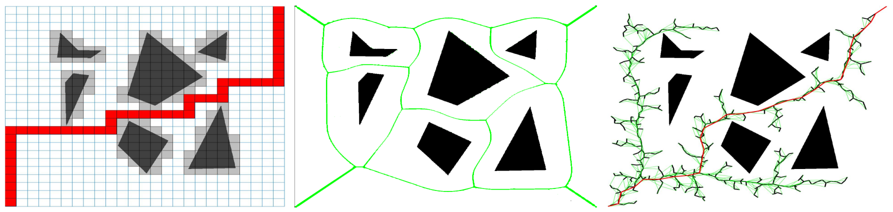

(a) Grid world and path. (b) Voronoi graph world. (c) RRT graph with path.

Fig. 2 Discretization schemes and plans generated from then. For the grid-based method (left), the plan follows connected discretized volume elements marked as empty. For the Voronoi method (middle), the plan connects to the nearest graph point, traverses the graph, then connects to the terminal point at its nearest graph point. The RRT (right) is like the Voronoi, but with the graph generated through a sampling process.

2.1 Planning in Discrete Spaces

Prevalent discretizations involve conversion of the world into a grid structure or a graph structure. Occupancy maps or occupancy grids are one common grid-based data structure. Ultimately, however, planning on grid-like structures gets reduced to planning on the equivalent graph structure with spatially determined connectivity (e.g., neighbor connectivity). One popular discrete planner is A* [4], a heuristic-informed extension of Dijkastra's algorithm [5] balancing depth versus breadth based search. If any part of the environment changes, the search must start over. The desire to admit re-planning from a previous invocation led to Dynamic A* (also known as D*) [6], which reduces the cost of replanning by performing local modifications of past searches. D* Lite [7] is a simpler and more efficient version, while Anytime D* (AD*) [8] is both anytime and incremental. Figure 2a depicts a grid-based planning scheme with the associated Manhattan world path, which uses only the 4 neighbors (up, down, left, right). Grid-based methods are constrained by the curse of dimensionality, whereby the search space grows exponentially with its dimension. Resolving fine details in a world requires exponentially more memory and results in longer planning times than using a coarser representation. The granularity of the grid implicity discretizes obstacle and robot dimensions, which may render feasible trajectories infeasible. Adaptive gridding structures and graphical processing unit (GPU) computation overcome the limitations of fixed grid world models and speed up planning time [9].

An alternative strategy to gridded world methods is to employ a sparse graph structure [10]. One such representation is the Voronoi graph of a world. The Generalized Voronoi Diagram (GVD) is a representation of free space consisting of the locus of points equidistant to obstacle boundaries, see Fig.2b. Graph search algorithms and heuristics can be used to find the shortest path through the GVD [11]. Given that the calculation of a GVD may be computationally prohibitive, an alternative graph creation method is Probabilistic Road Maps (PRM). The PRM algorithm constructs a graph of feasible paths during a preprocessing phase and uses this graph to find paths between desired points. It is probabilistically complete with established performance bounds [12, 2]. Lazy PRM decreases the runtime of the algorithm by only performing collision checks when searching for the shortest solution to a query [13], making the planner more suitable for single queries. PRMs have been extended for efficient replanning in dynamic environments by combining concepts from Lazy PRM and AD* [14, 15].

The PRM is an example of a sample-based planner where the sampling is done in advance. Instead of spending time generating the PRM, rapidly exploring random trees (RRT) are likewise probabilistically complete but perform minimal exploration, encouraging fast and efficient planning [16]. These properties are achieved by combining graph creation with plan search by building out the graph in a goal-directed manner through a combination of randomized sampling and heuristic or greedy (go-to-goal) sampling. An RRT planning instance is depicted in Fig.2c. Extensions to the basic RRT method include bi-directional search [17] and continued search to improve optimality [18]. Usually path smoothing is needed after identifying a solution due to the potential jaggedness of paths obtained from random steps. RRTs have been used for the path planning of autonomous vehicles [19]. They have also been extended for more efficient replanning in dynamic environments [20, 21, 22]. RRT-X is a recent extension that maintains and updates a single search graph throughout navigation and makes no distinction between local and global planning [23].

Motion planners can also be based on motion primitives. Sample-based planners that incorporate dynamic constraints include kino-dynamic planners, whereby connected nodes are reachable through a feasible control or constraint satisfying trajectory [24, 25, 26, 27]. In [28], such a planner permits aircraft to navigate through a dense forest. It uses one primitive to link any two points in space through constant control inputs and another to perform an aggressive turn. Since the primitives account for the dynamics of the aircraft, the planner only has to find a sequence of points for the aircraft to travel through and then use the appropriate primitives.

2.2 Planning in Continuous Spaces

Rather than converting the spatial representation of the world into a graph then inducing the same representation on the sought trajectories, one may seek to preserve the continuous structure of trajectories. If the control dynamics are ignored, then these strategies seek out continuous curves connecting the start and goal points. The simplest of such approaches is the *potential field method*, which employs potential functions to define a gradient-based vector field [29]. Following the potential field gradient as a differential equation yields the trajectory to follow. Since potential field methods have problems with local minima, extensions exist to arrive at formulations whose solutions are true global minima or have no local minima [30, 31, 32]. The same conversion applies to fast marching methods on a grid world [33, 34], which provide a more continuous trajectory versus the equivalent Dijkstra implementation over the underlying grid.

More complex approaches seek an actual trajectory by optimizing over the trajectory function space. The infinite-dimensional nature of trajectories requires finite-dimensional, parametric representations of curves to be used [35, 36]. If the actual control signals are also sought, then the problem becomes an optimal control problem, for which there are many solution strategies [37, 38, 39, 40, 41, 42, 38]. The main drawback to these methods is the computational cost associated with finding a feasible trajectory, especially when there are many obstacles. For computational efficiency, these methods are often iterative or gradient based in nature. They benefit from pre-conditioning the initial condition via a discrete planning method. If the overall problem has not varied significantly from the previous invocation, then the previous solution serves as a good initialization for iterative solvers [40].

A major issue for all global planners is the speed with which sensed data can be assimilated into the global map. For the slow global planners, global map updates are not the main bottleneck. However, for global planners that assert real-time operation, usually the cost to transfer sensor data to the internal representation and to update any important map data structures or cost functions far exceeds the plan update rate (by 1 to 3 orders of magnitude). Local planners exist to speed up this process by limiting what data is considered in the fast path updates. Following rapidly updated local plans gives the global planner time to generate an update based on the newly integrated information.

3 Local Planning

The need to augment global planning with local planning follows from the inability of global planners to re-plan at a rate compatible with the incoming sensory data and the underlying trajectory tracking controller. The purpose of the local planner is to generate real-time updates to the global plan, usually at the control feedback rate or close to it, in order to compensate for map errors, new objects, or moving objects. A common approach is to employ a hierarchical planner, in which the global and local levels use the same world representation though not necessarily the same data structure. Doing so means that a single method may serve two roles, however it requires that the conversion of sensory data to the world model occur faster than the sensory rate.

Early research on navigation primarily emphasized ultrasonic and laser scan methods since these sensors provided direct measurements of the external environment, as opposed to vision-based methods which required costly, at the time, processing to convert the raw visual signals into spatial data regarding the local environment. The dense 1D measurement signal provided from a laser scanner is sufficient for navigating through structured worlds where the scan plane provides correct information regarding collisions. With computing and sensory hardware improvements, navigation with dense 2D range or disparity image signals has started to become the norm. However, there are limitations to literally translating laser scan methods to admit dense imagery. Many of the existing solutions cannot scale with the data cardinality. This section covers the above history and topics.

Reactive Methods

Reactive methods use the locally sensed obstacle space and the current goal point to identify an immediately applied control policy. For ground vehicles, the control consists of a speed and steering command, though some methods assume a constant velocity model and simply steer. The potential field method [29] described earlier, due to its gradient following approach, is implementable as a reactive method. The Virtual Force Field method is one such implementation [43]. Sensor readings are used to update a global certainty grid of obstacles which is then used to calculate the repulsive forces of obstacles.

Rather than use point representations in a local Cartesian grid, the Vector Field Histogram (VFH) [44] method uses a local polar (histogram) representation for local points in the global map relative to the current robot pose. Processing of the polar histogram generates new steering commands, with an additional processing step to establish speed changes. The steering commands aim towards the best free-space option consistent with the current goal point. Improvements to the reactive policy include VFH* [45], which performs look-ahead verification using a short horizon forward time search with A*. It hypothesizes future polar histograms from the global map. Integration into a hierarchical planner for navigation was performed with VFH+ as the local policy [46]. The VFH approach was modified to apply to dense laser scanning sensors by processing directly on the sensor data in the Vector Polar Histogram approach (VPH) [47, 48]; it classifies travel directions into blocked and unblocked prior to determine the steering direction and the speed update.

Due to specific issues that arise from single policy reactive strategies, the Nearness Diagram Navigation (ND) method [49, 50, 51] first classifies the polar data into distinct environmental conditions. Each class is assigned a reactive, or sensor-driven feedback, policy for more robust and consistent navigation. The intent is to avoid the local minima and unstable motion sometimes exhibited by single policy reactive approaches. The ND navigation method factors the width of the robot into its reasoning. A hierarchical navigation implementation with global planning exists [52]. ND navigation is one of the early instances of gap-based navigation. Other gap-based approaches include [53, 54, 55] as well as [56] which considers the robot shape and kinematic constraints.

Velocity or Control Space Methods

The approaches considered up to now have not taken into account the vehicle's dynamics. Approaches operating in velocity space, in contrast, are able to accomodate nonholonomic kinematic constraints. For example, rather than look at instantaneous angles, or radial directions relative to the robot, the Steer Angle Field [57] algorithm evaluates forward integrated circular paths, discretized by steering angle. These sampled paths get checked for collisions, with the forward speed modulated by the existence–or lack thereof–of feasible paths. The method was integrated with a hierarchical navigation system [58]. Likewise, [59] performs local reactive obstacle avoidance by considering various circular trajectories and selecting one that maximizes the translational velocity, minimizes the angle to the target point, satisfies the robot's dynamics, and avoids collision for at least two seconds. A later implementation called the Dynamic Window Approach (DWA) [60] samples from a discrete set of relative control or velocity changes from the current control or velocity. It operates on a 2D occupancy grid model of the world and is capable of incorporating planar robot geometry into the collision checking. Like VFH, DWA has been extended to incorporate look ahead searching [61], best suited for dynamic obstacles. DWA methods exist which integrate with a global planner or into a hierarchical navigation strategy [62, 63].

A similar velocity space approach is Curvature Velocity Method (CVM) [64]. In this approach, obstacle avoidance is solved as a constrained optimization in velocity space. This allows speed and heading to be solved simultaneously and constraints to be added easily. These include robot velocity and acceleration as well as any application specific constraints (safety vs speed, etc.). The Lane Curvature Method (LCM) [65] addresses some shortcomings of CVM. It selects a *lane* based on collision-free distance and width and computes a local heading to guide the robot into the lane. CVM is then used to generate the necessary translational and rotational velocities. Since openings are chosen based on the width of the lane rather than angular width of the opening (as in VFH), the paths generated by LCM may be safer.

Optimal Trajectory Synthesis

Bridging the gap between reactive planners and velocity space planners are continuous trajectory synthesis implementations on the local map. On sufficiently small domains, they can run in real-time when capable of using the grid-based costmaps for cost and constraint evaluation. The Elastic Bands (EB) [66] planner generates a free-path (the elastic) in C-Space. This elastic deforms based on external repelling forces generated by obstacles as well as internal contracting forces for path length minimization. The EB approach has extensions for handling kinematic constraints [67]. The Timed-Elastic-Bands (TEB) [68] adds dynamic constraints. TEB has also been extended to maintain and optimize multiple candidate trajectories of distinct topologies [69].

3.1 Moving to 3D environment representation

A 2D environmental representation was sufficient when the primary sources of sensor data were planar, such as when using sonar, fixed (horizontal) laser scanning sensors, or other laser ranging methods [70, 71]. Dense sources of 3D points provide the opportunity to detect and avoid collisions with obstacles that planar sensors might miss, such as obstacles that protrude from the side or hang down from above. As a result, there is a motivation to utilize dense 3D sensor data when it is available. Common 3D dense data sources include Time-of-Flight (ToF) cameras, depth cameras, LIDAR, and triangulating scanners [72]. Stereo vision sensors also apply by simply generating a dense depth or disparity map.

One approach for utilizing 3D data with classical planning approaches is to project the data down to a 2D representation and run the planners as normal [73, 63], or performing a column-wise min operation over depth images for planar world information [74]. Generally, points are filtered in order to only consider those within the height of the robot. Though easy to implement and computationally cost efficient, there are some downsides to simplifying the environmental representation in this way. Unless the robot's cross sectional geometry is constant with height, the simplified representation will be overly conservative; valid configurations may be detected as in collision with the environment.

Another approach is to maintain a 3D world model and use this for planning. However, there are several challenges with doing this. First, not all local planning approaches are readily adaptable in this way. For example, gap aiming (directional) approaches [53] are explicitly designed to process a 1D list of measurements (such as from a laser scan). Indeed, only recently have gap aiming approaches explicitly considered non-point, non-holonomic robots [56]. Sampling-based approaches, on the other hand, have an explicit trajectory scoring step which can be modified to perform collision checking against a 3D environmental representation. Such a tactic is taken with discrete steering directions checked against a filtered point cloud model of the world [75]. Likewise [76] compares a fixed set of sample trajectories against a local point cloud. Using stereo the aerial robot detects points at a given range, and propogates them in time for a richer world model. The range-filtered measurements keep the point cloud model sparse enough for real-time operation on lightweight computing platforms.

Another challenge is selecting the environmental representation to use. One of the simplest is the voxel grid (a 3D cartesian occupancy grid), however memory requirements for voxel grids are high due to the curse of dimensionality. Variable resolution structures can significantly reduce the amount of memory required using datastructures such as octrees [77]. For aerial vehicles, the volumetric map can be height filtered for more efficient processing [78]. Rather than employ efficient occupancy data structures, efforts have considered alternative point cloud data structures with efficient query times. Points can be stored in sorted datastructures such as kd-trees to allow faster nearest-neighbor queries [79]. The method is very efficient when the sensor data is of low cardinality, the local volume is restricted, and the robot model is a point. There is an added computational cost to build and maintain these data structures.

Even if the planning algorithm itself could operate in real time, the process of updating the world representation introduces latency between sensing and planning. An alternative approach is to avoid the reconstruction process and perform planning in an earlier visual representation. For navigation strategies using stereo camera systems, this involves planning directly in disparity space [80, 81, 82] or planning using combinations of optical flow and stereo disparity [83, 84]. After gathering dense data from sensors, these approches explore C-Space trajectory sampling and collision checking in the perception space. Local path options are mapped into the image space for evaluation [85, 86], where the objective is to follow the path generated by a global planner such as A* while avoiding unmodeled obstacles. Perception space approaches improve time performance by reducing the delays associated with populating and maintaining in-memory data structures.

4 Neuroscience Research Related to Navigation

The compartmentalized structure of robotic navigation algorithms is likewise reflected in the human brain. Though different from engineered systems, cognitive and behavioral neuroscientists and researchers have identified distinct processing regions and characteristics regarding navigation via controlled experiments [87, 88, 89, 90, 91, 92, 93]. The findings suggest that perception for navigation involves both egocentric (or viewer-centric) models for decision-making, as well as object-centric (or world-centric) models, sometimes also referred to as allocentric. The former corresponds to visual understanding relative to the human's reference frame, whereas the latter is relative to an external reference frame usually belonging to an object receiving attention. Similar structural differences in computation have been proposed by Marr when describing a representational framework for vision [94] and its computational aspects. In particular, from low-level to high-level, he described four primary components as depicted in Fig.3. The first is the raw information associated to the sensed scene, the input image. The second, called the primal sketch, consists of basic signal processing transformations of this image that extract geomoetric and volumetric primitives from the image proper (lines, areas, blobs, etc.). The third, called the 2.5D sketch, reflects higher level interpretation and analysis of the image and its low-level features. The representation of information is still at the image level, however depth data or depth-dependent information exists as a layered representation. The layered images contain information about the distal ordering of the scene relative to the viewer. Represented internally using image-based data structures, these three initial levels are viewer-centric. The last layer reconstructs or generates a richer 3D model of the objects sensed within the scene. At this level the reference frame or viewpoint shifts away from the viewer and to a global reference frame or an object-centered reference frame. In this manner, the

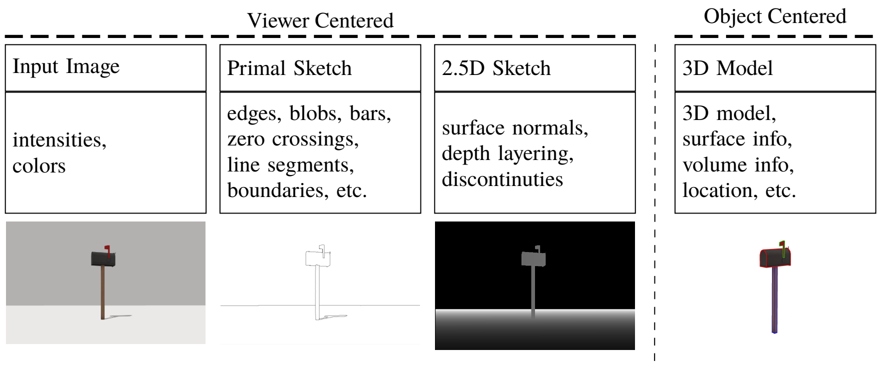

Viewer Centered
Object Centered
Input Image
intensities, colors
Primal Sketch
edges, blobs, bars, zero crossings, line segments, boundaries, etc.
2.5D Sketch
surface normals, depth layering, discontinuties
3D Model
3D model, surface info, volume info, location, etc.

Fig. 3 Marr's representational framework for visual processing. Of the four stages depicted, three are in viewer centered representations, with the fourth in an object centered representation.

reconstructed model is independent of the user's view and may persist as it changes in response to relative motion between the viewer and the object. Naturally, missing from this description is the agent's memory of past scenes or locations. However, it is sensible to imagine that both egocentric and allocentric memory and predictive models exist [87, 88].

Though distinct regions and reference frame processing paradigms have been discovered, there is still coupling between them. The research suggests that early computation may primarily rely on egocentric models with some influence from allocentric models [90, 91]. Likewise, information related to egocentric navigation appears to be important for detecting or estimating important allocentric states or collision informing properties [87, 95]. Thus, within the neural processing hierarchy, egocentric processing happens earlier than allocentric processing but can be influenced by earlier allocentric outcomes expected to persist into the near future. With regards to objects, Marr's framework predicts that object information would primarily be encoded allocentrically. However, since evidence for an object arises from lower level evidence, we should expect that some information regarding objects exists in an egocentric representation. There is indeed evidence that egocentric representations are used for object locations [96] as well as for navigation goal states [97]. There is also support for brain regions that translate between the two representations, such as from egocentric to allocentric representations (ex: what is my current position on this map given my current view) and vice versa (ex: in order to head west, I need to turn left) [98]. Rat studies have shown that the same stimuli will affect different regions based on their egocentric or allocentric properties [99].

The research suggests that both representational forms for modeling the world should exist within a navigation pipeline. In particular, methods employing only world-centric models at all levels of the navigation hierarchy may be misguided. Yet many navigation frameworks are characterized by this property. Instead, viewer-centric models should be incorporated and prioritized within components requiring fast decision-making and low latency from the sensory input to the controlled response. Additionaly, world-centric models should be prioritized for slower decision-making and reasoning that operates beyond the local frame or rquires an external frame to simplify processing. Furthermore, the two processes should be coupled at a rate consistent with navigational decision-making.

5 PiPS: An Ego-Centric Navigation Framework

This section introduces and expands upon a local planning method employing perception space (PiPS) [100]. In relation to Marr's visual framework, PiPS operates in the mid-level 2.5D representation whereas it is assumed that the global planning method will operate in the world representation. This section first covers the structure of the navigation system, including general details on how the global and local planners integrate. It then covers the collision checking process of PiPS and compares the time costs of various data structures used for collision checking against local 3D environment models. Extensions to PiPS for integration into a hierarchical navigation system are covered in the remaining sections. In particular, the original memoryless PiPS is augmented both with memory and a local cost-map component. These augmented data structures are called the egocylinder and egocircle. They provide means to perform collision checking and trajectory scoring, respectively. The final integrated, PiPS-based navigation system results in a mixed representation navigation scheme with linear scaling properties in the data cardinality and real-time obstacle avoidance properties. Navigation simulations and visualizations are created using Gazebo and RViz.

Navigation using Move Base.

The ROS Move Base package provides an API for hierarchical goal directed navigation of planar robots. Its design enables the incorporation of different global and local planners. These are connected by a trajectory synthesis module and several scoring systems that evaluate or influence the final local trajectory chosen. Move Base, like many other navigation frameworks, operates under the assumption that the global and local planning representations will be the same, e.g., grid-based. As a consequence, the scoring mechanisms fundamentally rely on *costmaps*.

Costmaps are 2D grids populated with travel cost or other scalar value for assigning a cost to each grid cell. These costs inform the Move Base planners. There are several different sources for the costs, with one being the occupancy grid. An occupancy grid keeps track of navigable space and non-navigable space (or occupied space) associated to obstacles. Obstacles are added to the occupancy grid from sensor data such as laser scans or point clouds. Grid points between the sensor location and detected obstacles are marked as unocuppied via ray tracing. To factor in the robot's radius, inflating each occupied cell through dilation techniques generates an inflated occupancy grid (usually by the smaller radius if the robot is not circular). Generating the obstacle distance map involves assigning each unoccupied cell the distance to the nearest occupied grid cell. This obstacle distance map informs the creation of path scoring costmaps. There is a global costmap used for global planning. It covers the entire region of known space within which planning occurs. If a prior model for the environment is available, it can be used to initialize the global costmap. Otherwise, the entire costmap is initialized as unknown and considered traversable for the purposes of global planning. In addition, there is a local occupancy grid that tracks occupancy information in a robot centered local Cartesian grid whose orientation is fixed relative to the initial orientation of the robot. Figure 4 depicts such a local map where the initial robot orientation differs from the current robot orientation and the world orientation, hence the rotated projection of the local map onto the world. The local nature of the grid means that costmaps generated from the local occupancy grid are only defined for a fixed Cartesian domain centered on the origin. The domain dimensions are set to be comparable in size to the sensing region of the robot (though they could be set smaller if desired). The local costmap is key to local planning and must update faster than the sensor rate to admit the real-time synthesis of new local paths. Figure 5 depicts key costmaps associated to local planning. Low cost regions are red, while high cost regions tend to blue/purple. The obstacle cost penalizes robot trajectories that pass too close to obstacles. The local goal cost penalizes sample robot pose locations based on the minimum travel distance to the local goal point, which is the point of the global trajectory (marked as a black curve in the costmaps) at the boundary of the local map. Basically, it penalizes trajectories for ending away from the local goal. The path cost penalizes robot pose points based on their distance to the nearest global path point. Higher costs are associated to trajectories straying from the global path. The time to populate these costs from the occupancy grid varies from $O(n)$ to $O(n \log n)$ where n is the grid area [33, 101].

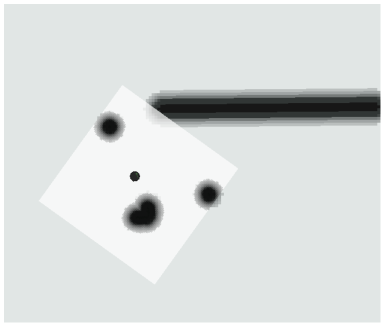


Fig. 4 Mobile robot navigation with visualization of the local map as a square centered on the robot.

The global planner in Move Base uses a variant of Dijkstra's algorithm to find a path from the robot's current pose to the specified goal, while the local planner generates velocity commands to direct the robot along this path. The default local planner provided with Move Base is the DWA local planner (DLP), which invokes DWA [60] to sample velocity commands. Sampled velocities are forward simulated to create trajectories. Each trajectory is scored based on a weighted sum of costs for the described cost functions: the obstacle cost, the goal cost, and the path cost. The local and global planners run at specified frequencies. If the local planner fails to find a valid velocity command, global replanning is triggered. If the local or global planner fails for longer than its specified time limit, a recovery behavior rotates the robot in an effort to clear obstacles from the local costmap. If all recovery behaviors have run and the planner is still unsucessful, navigation aborts.

PiPS Modifications to Move Base.

The principal components of Move Base are depicted in the data flow diagram of Fig.6. Not depicted is the robot pose information generated by the odometry or localization process, which is required to propogate forward in time past measurements. This estimated pose update will be denoted g_{move}. The traditional DLP implementation involves the blue blocks, the black blocks, and the dashed block. We will modify the local planner to employ an egocentric representation for the world based on distal information of obstacles and global path points and maintained in a 2D or 1D array, thereby operating in perception space rather than in world space. These are the red blocks labeled Ego-Circle and Ego-Cylinder, which correspond to the propagation of perceptual memory regarding the local egocentric representation of occupied space. Additional important modules include collision checking and path scoring, which are both part of the local planner. Employing a mixed representation requires extensive modification of the existing scoring methods, especially those linking the local path to the global path.

5.1 Collision Checking in Perception Space

Rather than map the sensor data regarding the local world into a world centric represention, PiPS keeps the sensor data in a viewer centric representation. Collision checking requires mapping the robot model into the same viewer centric representation and comparing the robot model's distal properties to those of the surrounding environment. Thus, rather than evaluating collisions using 2D or 3D world maps, we

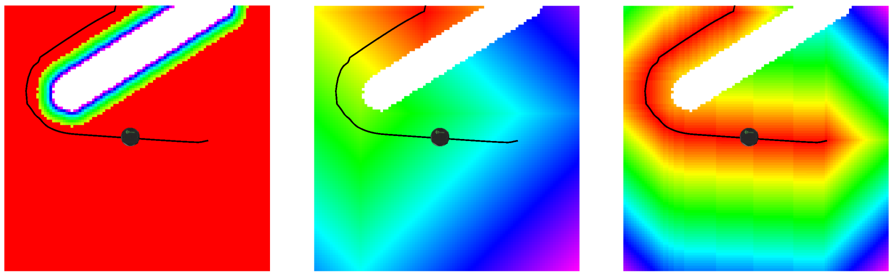

(a) Obstacle cost. (b) Local goal cost. (c) Path cost.

Fig. 5 Visualization of the trajectory scoring as cost maps computed on an occupancy grid. Low cost is red and high cost is blue/purple. The black curve is the global path to follow.

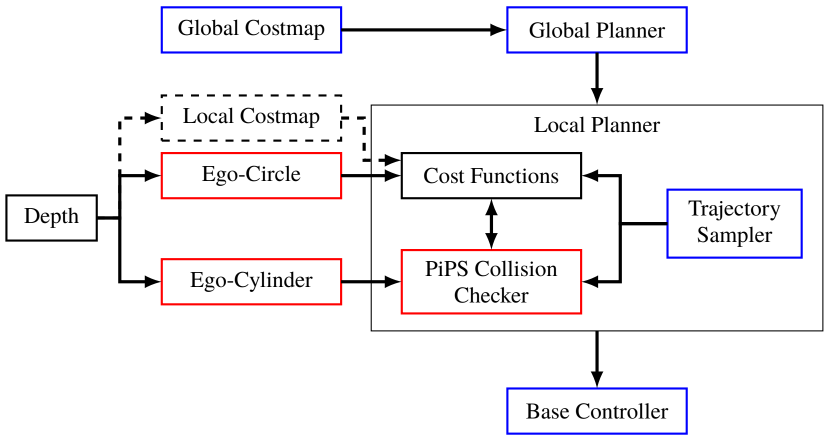


Fig. 6 Block diagram depicting processing and data flow for the PiPS-based navigation strategy. Design considerations for the critical components are covered in this section. The blue components are unchanged, the red components are PiPS enhancements. The dashed component is only used by the traditional Move Base pipeline. The "Cost Functions" block is also modified for PiPS.

evaluate collisions using 2D images. Collisions occur when the robot model maps to depth layers that lie further away from sensed world depth layers in a given region of space. In the opposite case, if the robot geometry of a test pose maps to image regions whose obstacle depth layers are further away than the robot depth layers, then the test pose is considered safe. In this manner, PiPS switches the main collision-checking computations from relying on the transformation of sensory data to world data for collision checking of the robot in the world, to relying on the transformation of the robot world model to the sensor representation for collision checking in image-space.

Mapping of the robot to the sensor representation requires modifying the graphics z-buffer rendering pipeline. To render an object model, the traditional approach to graphics considers all object points that project to the image then chooses at each pixel only the closest point projecting to it as the measurement source. The remaining points that project to the camera at that pixel are ignored. This is a closest point model. Given closed mesh models of all objects in the world, the set of projection points that may realistically project to the camera are those whose outward normal to the mesh points toward the camera. For collision checking we wish to synthesize not the closest point to the camera, but the furthest point from the camera (with a normal vector pointing in agreement with the camera optical axis vector).

Given a surface mesh model of the robot, synthesis of the collision depth image of a hallucinated robot in an empty world will select the furthest points. Let the set $\left\{ (q^i, \hat{n}^i) \right\}$ of robot points with associated outward surface normals be those that project to the camera at pixel r. Then the chosen point to be used for the synthesized depth image is the one with the index

$$i^* = \arg\max_i D(q^i) \quad \text{subject to} \begin{bmatrix} \hat{n}^i \\ 0 \end{bmatrix} \cdot q^i > 0 \wedge z > 0, \tag{1}$$

where $D : \mathbb{R}^3 \rightarrow \mathbb{R}^+$ maps points to depths. The depth of point q^{i^*} is placed at pixel r. All points in the image of the hallucinated robot that do not have a projected point are simply set to 0.

As the hallucinated robot pose $g \in SE(3)$ changes, the collection of surface points and normals will change under g, leading to different depth images. This robot pose is in collision with actual objects in the real world if the sensor image D_m has a value less than that of the hallucinated robot image $D_H = D \circ g(\mathcal{M}^{\mathcal{R}})$, where $\mathcal{M}^{\mathcal{R}}$ is the mesh model of the robot (simplified to a set of points and outward normals). Simply put, a hallucinated robot pose is collision-free if

$$D_m(r) > D_H(r) \;\; \forall r \in \mathcal{I}, \qquad (2)$$

where $\mathcal{I}$ is the image coordinate domain. A collision-free, hallucinated robot pose is called a *safe* pose.

Figure 7 depicts the process just described. Viewing from left to right, the top row consists of the world view for a pose testing scenario. The real robot (black) contemplates itself at a future trajectory pose (red) that is not safe. Collision checking is not performed with the more complex true model, but rather with a cylindrical simplification (red cylinder in second column). For collision purposes in depth space, only the far sides of the robot matter (cyan surface in third column). A portion of the robot collides with walls in the real world (yellow surface in fourth column). The blue volume depicts the projection cone associated to the tested robot pose. Image depth values indicating world point within this cone indicate a robot collision at the hallucinated robot pose. PiPS does not perform the collision checking from the world (or third person perspective). Rather it performs the collision checking in the viewer-centric perception space (or from the first person perspective) as depicted in the second row. From left to right, the first person views are visualized. The right-

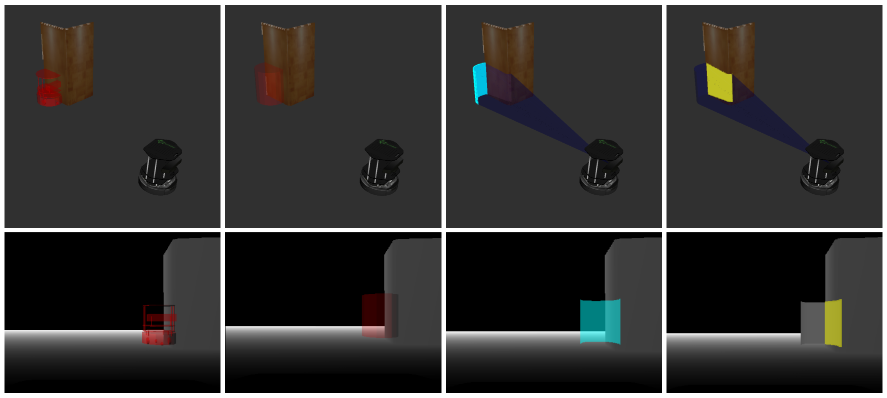


Fig. 7 Each column illustrates a conceptual step of PiPS. The top image depicts the scene from a third person perspective, while the bottom image depicts it from the robot's first person perspective. From left to right: hallucinate the robot at a pose (red); Replace the robot with a simplified geometric representation (red); Find the far surface of the robot (light blue); Detect collisions (yellow)

most column actually involves the overlay of the two depth images, as can be seen by the gray depth values corresponding to the robots far cylindrical surface patch. The yellow region contains robot depth values that are greater than the measurement depth image values.

Efficiency of Perception-Space Collision Checking

The PiPS local planner in [100] sampled from a pre-determined set of trajectories that remained in the field of view relative to the robot's current perspective, then selected the longest non-colliding trajectory. It was capable of real-time operation on embedded processors with the computational power of *circa* 2014 cell phones, thereby demonstrating favorable processing properties. It can run in real-time on today's embedded system-on-a-chip processors, such as those using the latest Arm Cortex chips. Here, we explore more deeply the computational cost of PiPS. The asserted value of a perception space approach to collision checking is that conversion of the sensory data to alternative world representations incurs a time cost that determines the minimal response latency of the local navigation policy. This section performs a collision check time cost comparison for several world model data structures and collision query strategies. The data structures chosen for comparison are depth image (PiPS), point cloud, octree, and k-d tree. With reference to Fig.6, the latter data structures would require swapping out the "PiPS Collision Checker" block with the appropriate collision checking implementation, and would involve programming alternatives to the Ego-Circle and Ego-Cylinder blocks. The first evaluation metric of interest is the time cost of initializing the data structure from a new sensor input. This value is significant because it represents the minimum latency between receiving environmental information and being able to act on it. The second evaluation metric is the time cost of performing a collision check with a given robot pose. Together, these impact the time cost of collision-checking a single trajectory or a set of trajectories. The timing experiments were performed on an Intel Xeon E5-2640 @ 2.50GHz processor with single-thread Passmark score of 1468 and multi-threaded score of 9512. All reported timings are for single-threaded collision checks.

The investigated data structures require different amounts of processing to initialize from sensor data. For the PiPS depth image process, the only processing performed is transposition of the incoming depth image for more efficient pixel-wise comparisons during collision checking. The point cloud approach requires converting the incoming point cloud sensor message to a Point Cloud Library (PCL)[102] data structure. The k-d tree and octree approaches also require the input depth image to be converted to a point cloud for populating the data structures. The k-d tree implementation is from the Point Cloud Library (PCL) [102]. The implementation used for the octree is octomap [77]. An octree with 5cm resolution is created from a point cloud using code derived from the Octomap Server ROS package [103].

The time required to perform the preparatory conversions is listed in Table 1. Conversions are performed using nodelets from the *depthimage_to_laserscan*,

Table 1 Average times (in ms) for data preparation/conversion to necessary input format.

Resolution:	640x480	320x240	160x120
Depth image to PointCloud	2.2	.61	.21
Depth image to LaserScan	.66	.35	.22
Decimate	.084	.31	.20

Table 2 Average times for initializing datastructures from an input (in ms).

Resolution	640x480	320x240	160x120
Depth image	.64	.14	.048
Point cloud	6.67	2.74	.59
K-d tree	34.0	7.8	3.7
Octree	245	88	42.7

Table 3 Average times for collision checking a pose (μs).

Resolution	640x480	320x240	160x120
Depth image	55.0	31.6	19.6
Point cloud	882	450	118
K-d tree	8.9	7.4	6.9
Octree	11.4	11.6	11.6

depth_image_proc, and *image_proc* packages. Depth images from a ROS/Gazebo simulated environment pass through the Decimation nodelet and on to the PointCloud and LaserScan nodelets. Custom timing nodelets record the time required for conversion and are enabled for only one conversion nodelet at a time. Measurements are collected until the average value is stable to at least 2 significant digits.

Across the board, the depth image decimation time remains lower than the other conversion strategies. The non-zero cost of the 640x480 decimation approach represents the overhead of passing an image on without any processing.

We also measured the average time required to populate each data structure from new sensor inputs of several different resolutions and report the results in Table 2. The per-frame and per-pose computation cost is calculated using a collection of 791 depth images captured while the robot wandered through a ROS/Gazebo simulated environement. A set of 200 poses, representing a set of trajectories, was also generated. In order to evaluate how computational requirements scale with the size of sensor data, each set of tests is repeated with the depth image decimated to the following sizes 640x480, 640x240, 320x240, 320x120, 160x120. All of the collision checkers are tested using the exact same series of sensor data and candidate poses. The collision checkers are evaluated offline one after the other, each testing all poses on all images.

The table demonstrates that the data structures have poor scaling properties when increasing the amount of data to process. Real-time operation requires significant decimation of the input data, thereby losing important structural knowledge of the local environment. Though the decimation enables real-time processing for small compute platforms as would be deployed on weight restricted robots, like quadcopters, it does not scale well for mobile robots without these restrictions. At full resolution, the time cost of conversion can exceed the data arrival times for sensors operating at typical frame rates (presumed to be around 30Hz).

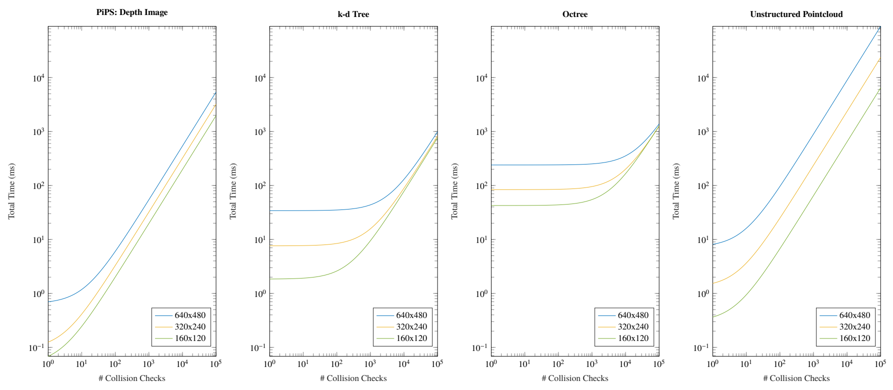


Fig. 8 Semilog plots of the total time to initialize data structures with one sensor input and to perform N collision checks. From left to right, the methods tested are depth image (PiPS), kd-tree, octree, and point cloud.

The next performance metric is the average time required to collision check a single candidate pose of a robot. All else being equal, a smaller value will allow more poses to be tested in a given time frame. For the purposes of these tests, we assume a cylindrical robot. Depth image collision checking uses the approach described in §5.1. The point cloud approach naively loops through all points, checking if any lie inside the specified cylindrical region. The k-d tree approach first queries the tree for any points lying within a sphere bounding the candidate robot pose cylinder and then checks if any of these are within the cylinder. The octree approach uses the Flexible Collision Library (FCL) [104] to check for collisions between the populated octomap and a cylinder. We measure the average times required to collision check a candidate robot pose for several candidate resolutions and report the results in Table 3. The table demonstrates why some researchers choose to employ specialized data structures, as their collision checking time cost can be quite low and nearly constant versus image resolution. Once the data structure has been populated, collision checking is the lowest cost and practically negligible in comparison to the preparation time. The PiPS depth image approach, on the other hand, has a larger resolution dependent time cost. The value of a PiPS approach lies in the total cost.

The results in the Tables indicate that using the depth image approach can result in significant time savings, so long as the number of collisions tested can be kept low enough. While the per-collision check time of the depth image approach is not as low as that of the k-d tree, the initialization time of the depth image data structure is much lower than that of the k-d tree. For example, with 640x480 sensor data, a depth image approach can initialize its data structure and then perform over 600 collision checks in the time it takes the k-d tree to just initialize. The graphs in Fig.8 compare the total time for initializing the data structure and performing a given number of collision checks across the different approaches. Both axes have log spacing. The value of the k-d tree and octree approaches lies in the near flat curves for less than 100 collision checks, and the relatively low slopes after that. In contrast, the PiPS depth image approach has a mostly linear curve, just like the point cloud approach, albeit with an improved base cost. When the expected number of collision checks is relatively low, the depth image approach is significantly faster. Identifying where the depth image curves cross a comparison strategy indicates how many collision checks should be performed to achieve an equal compute time. Similar findings should hold for robots with different geometries, however the slopes may differ.

Importantly, during actual deployment, the robot will not be evaluating new trajectories at every frame. Rather, the first step in the process is to test the current local path for feasibility. If it is feasible, then the navigation system continues to drive along the local path. New path sampling, scoring, and testing happen when an obstacle is detected along the current local path or the robot nears the terminal point of the given path. Evaluating the time cost to collision check the current path, it is clear that the PiPS approach has a significant advantage as it can perform this check and determine the feasibility of the current path before any of the other methods can even initialize, even if the comparison is between a full resolution PiPS implementation and most decimation levels for the alternative implementations. The time cost for PiPS to perform this check is 6.2ms, 3.6ms, and 1.3ms across the resolutions evaluated, for the case that the number of robot poses to test in the current trajectory is 100. This time will decrease as the robot moves along the current trajectory.

5.2 Egocylindrical Perception Space for Enhanced Awareness

Though navigating with PiPS using trajectories that map into the current sensing image domain is collision free [100], the published approach employs simple straight line trajectories. Except for the floor space immediately in front of the robot, the sampled trajectories map the robot into the depth image. However, visual sensors have a limited field of view when compared to laser scanners (usually around 60-90 degrees horizontally versus 270-330 degrees horizontally), thus restricting trajectories to the field-of-view is quite limiting. DWA and many other local planners sample a richer trajectory space whereby some trajectories leave the visual sensor's FOV. Not factoring in these trajectory segments for trajectory scoring or collision checking leads to unsafe navigation. Much like a local cost-map accumulates and propagates occupied points, the local PiPS module requires the ability to accumulate, propagate, and retain previously seen perceptual information lying outside of the current FOV. This will be done using the egoclindrical image space [105], another 2.5D image space representation whose theoretical domain extents surround the robot. Due to the nature of camera projection equations, which require positive z-values in the camera frame, traditional depth images cannot retain information of world geometry behind the robot. Furthermore, the homographic projection involved in traditional pinhole models requires an infinite image region to map the forward-facing half-plane to an image. The egocylindrical perception-space representation avoids these modeling degeneracies. World points from sensors are projected onto a virtual cylinder surrounding the robot and are propagated as the robot moves. The surface of the cylinder is discretized into a 2D grid.

Whereas the egocylindrical image in [105] stores stereo disparity values, the egocylindrical image here stores the ranges corresponding to each point on the virtual cylinder. In relation to the previous section, §5.1, the only modification required is on the image domain and the projection equations for rendering the depth image measurement (now as an egocylindrical image). The left side of Fig.9 visualizes the egocylindrical (image or perceptual) representation. The color coding indicates distance of world points from the sensor's optical origin. The right side provides a third person view of the scenario, simulated using ROS/Gazebo. The simulated sensor in this scenario has a forward facing field of view of 60 degrees.

When the sensor is a depth sensor, then the pixel depth data is based on the ray projecting out from that pixel. Mapping the depth value to a range value requires factoring in the ray information. Using the homogeneous image ray representation with unit z-coordinate, the egocylindrical range is

$$\rho = D_m(r_{im}) \left\| \begin{bmatrix} x_{ray}(r_{im}) \\ y_{ray}(r_{im}) \\ 1 \end{bmatrix} \right\| = D_m(r_{im}) \rho_{ray}(r_{im}), \tag{3}$$

where (x_{ray}, y_{ray}) are the important ray coordinates obtained from the image pixel coordinates r_{im}, and ρ_{ray} is the corresponding length of the ray when treated as a vector. For efficiency purposes, the function values $\rho_{ray} : \mathcal{I} \mapsto \mathbb{R}^{+}$ should be precomputed and stored in an image whose dimension is the same as the depth image for direct lookup. Note that the camera convention is for the z-coordinate to point along the optical axis and for the x-coordinate to be horizontal with the y-coordinate pointing downwards.

The mapping of the depth value to the egocylinder coordinates involves computing the angle coordinate θ and the height value z_{cyl}, as per

$$\begin{aligned} \theta &= \operatorname{Arg}(x_{ray}(r_{im}) + j) = \theta_{im}(r_{im}) \\ z_{cyl} &= D_m(r_{im}) y_{ray}(r_{im}) \end{aligned} \tag{4}$$

where the constant imaginary term j is due to the unit z-coordinate in the ray representation. As above, for computational efficiency, the $\theta_{im} : \mathcal{I} \mapsto \mathbb{R}$ and $y_{im} : \mathcal{I} \mapsto \mathbb{R}$ functions should be precomputed over the image domain. If the sensor is a range sensor, the ray will have unit length rather than a unit z-coordinate. The appropriate modifications of the above equations will be needed.

These points then get mapped to egocylinder image coordinates $r_{cyl} \in \mathcal{I}_{cyl}$ using the homogeneous egocylinder projection matrix K_{cyl},

$$r_{cyl} = K_{cyl} \left[\begin{array}{c} \theta \\ z_{cyl} \\ \hline 1 \end{array} \right] \qquad \text{where} \qquad K_{cyl} = \left[\begin{array}{cc|c} f_h & 0 & h_c \\ 0 & f_v & v_c \end{array} \right], \tag{5}$$

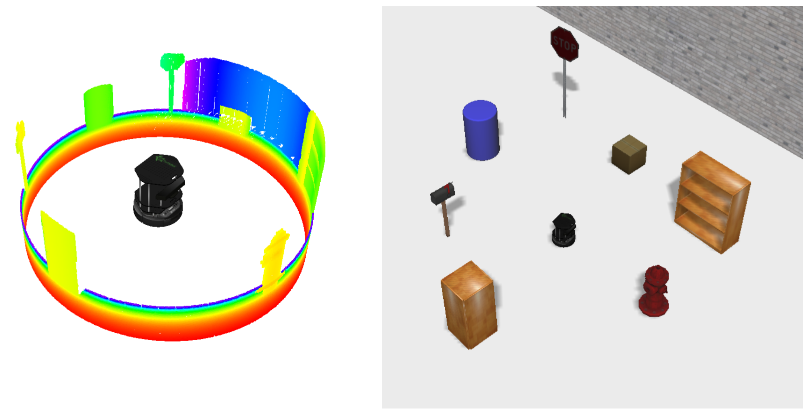


Fig. 9 Gazebo/RViz visualization of virtual egocylinder (left) and the environment it represents. Range colormapped from near to far goes from red to blue/purple.

for $f_h = f_v = \cot(2\pi/n_{cols})$, $h_c = n_{cols}/2$, and $v_c = n_{rows}/2$, where $n_{rows} \times n_{cols}$ are the egocylinder image dimensions (note that v_c can be shifted if the domain is biased upwards or downwards). To identify the appropriate bin to map the point into, the decimal coordinates r_{cyl} should be discretized to whole numbers. The Cartesian coordinates of the point are stored in the bin $\mathcal{B} = \mathcal{B}_{ego}(r_{cyl})$ contained at the discretized coordinate location $\lfloor r_{cyl} \rfloor$, where $\mathcal{B}_{ego}$ consists of all bins in the egocylindrical map representation indexed by discretized coordinate locations.

To generate a range map using the egocylindrical representation, it suffices to compute the range of the point in each bin. For shorthand, we write

$$D_m(r_{cyl}) = \rho(\mathcal{B}_{ego}(r_{cyl})) \tag{6}$$

which then renders an egocylindrical range image of all points stored in memory. When updating the egocylindrical representation with recently sensed depth or range information, the new data overwrites the stored data.

Synthesis of a hallucinated egocylindrical image employs the egocylindrical projection equations instead of the standard pinhole projection equations. Collision checks involve the same conceptual procedure described in §5.1, but with egocylindrical range values instead of depth values. The time cost to perform collision checking with egocylindrical images is close to that of traditional depth images, thus the egocylindrical collision time cost curves resemble those of Fig.8.

Egocylinder Propogation

Propagation of the stored points involves transforming them under the motion induced pose change $g_{move} \in SE(2) \subset SE(3)$ from one time point to the next, where g_{move} gives the coordinate frame of the old robot pose relative to the new robot pose. Define the egocylindrical coordinate vector $p_{cyl} = (\rho, \theta, z_{cyl})^T$ and the Cartesian coordinate vector $p = (x, y, z)^T$. Consider the mapping from egocylindrical coordinates to Cartesian coordinates T_{e2c} and vice-versa T_{c2e},

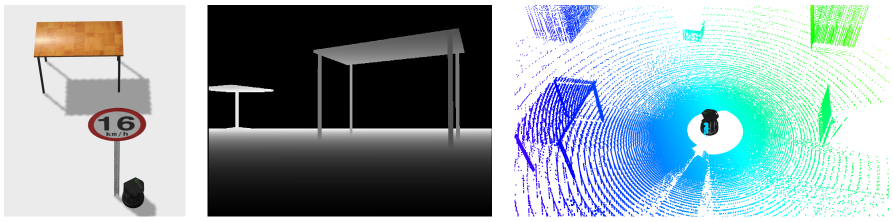
$$p = T_{e2c}(p_{cyl}) = \begin{bmatrix} \rho\cos(\theta) \\ \rho\sin(\theta) \\ z_{cyl} \end{bmatrix} \quad \text{and} \quad p_{cyl} = T_{e2c}(p) = \begin{bmatrix} \sqrt{x^2 + z^2} \\ \mathrm{Arg}(x + jz) \\ y \end{bmatrix}, \qquad (7)$$

both with reference to the viewer/camera frame. The new egocylindrical coordinates p'_{cyl} of a stored point p_{cyl} are:

$$p'_{cyl} = T_{c2e} \circ g_{move} \circ T_{e2c}(p_{cyl}). \qquad (8)$$

Identifying the new bin that the point should be moved to involves applying the projection matrix K_{cyl} to the last two coordinates in p_{cyl} mapped to homogeneous form, then discretizing the resulting coordinate outputs. In the event that multiple points map to the same bin, only the point with the lowest range is kept. To speed up calculations, our implementation only keeps track of p, using p_{cyl} solely to determine in which bin to store p.

With propagated and depth image updated egocylindrical data, synthesis of the egocylindrical range image contains historical knowledge regarding the local environment, thereby mitigating the FOV issues of depth images. The egocylindrical image enhances collision-checking and collision-free navigation when performing tight cornering and maneuvering around obstacles. Figure 10 depicts a scenario whereby the mobile robot is close to a signpost and with the signpost outside of the field of view, as noted by the lack of a signpost in the depth camera image. However, the egocylindrical data structure does contain points from the signpost's pole. It is the small blue point cloud near the robot in Fig.10c. The point cloud was generated from the egocylindrical data (using T_{e2c}). The egocylindrical representation is not guaranteed to be correct, as world geometry that was never seen does not exist in the model and is not propagated. Thus it is possible to conclude that a trajectory is safe though it may not be. Forward navigating mobile robots usually exhibit this



(a) Gazebo World (b) Depth Camera (c) Egocylinder Points

Fig. 10 Visualization of an out-of-FOV obstacle scenario. The robot needs to turn right to avoid colliding with the signpost, which is currently outside the FOV of the depth sensor. Since the signpost was visible to the depth sensor previously and is in the egocylinder, it can be avoided.

problem at the beginning of a global trajectory and less so later on due to the fact that forward travel will propagate the seen world out into the unseen portions of the egocylindrical image. The main danger lies when performing high angle turning into unsensed world regions. The role of the global path and the local planner scoring functions is to prevent these situations from happening by giving preference to safer trajectories.

5.3 Egocircular Representation and Trajectory Scoring

The egocylindrical perception space representation provides an efficient viewer-centric means to collision check based on current and previous perception space measurements (e.g., depth, range, or disparity). However, it is not an efficient means to score trajectories for collision assessment purposes. The relatively high slope of the PiPS collision-checking cost means that only a small set of trajectories should be tested for collision checking. Thus, the typically large set of trajectories sampled in sample-based methods should be rank ordered with only the top few being evaluated. Additionally, the PiPS approach can only give an indication of safe or unsafe. It cannot score based on proximity to obstacles or other pertinent geometry or goal information. In traditional local planners, the transformation of sensor-based obstacle geometry to occupancy grids occurs because of the ease with which distance or proximity information can be generated (though there is a significant time cost if the occupancy grid is 3D). The distance information is essential to scoring and rank ordering the sampled trajectories.

For fast trajectory scoring, a more compact and efficient representation of the local collision space is necessary. For that we employ an *egocircle*, which can be thought of as a flattening of the egocylinder image model to a 1D space or laser scanner type of space; the true calculations will be different but the conceptual idea is correct. The egocircle is an egocentric polar obstacle data structure reminiscent of the data structure used in polar based methods [44, 45, 47, 48]. Its purpose is to populate, propagate, and store the local environmental history necessary for approximately and efficiently scoring candidate trajectories relative to obstacle proximity, goal point proximity, and global path following. These scores provide ranked orderings of the sampled trajectories.

The local planner block depicted in Fig.6 initially samples a rich set of trajectories, scores them according to predetermined criteria using the egocircle data, then collision checks them according to their score ranking using the egocylindrical representation and PiPS collision-checking. The first sample to pass the collision-check module is the trajectory to follow for the next local planning period. Because collision checking occurs using the egocylindrical representation, the egocircle scoring does not need to be a strict or conservative scoring method. Rather it can be liberal and admit collision inducing trajectories. Its design is meant to provide efficient scoring, data storage, and propagation implementations.

Egocircle Measurements, Storage, and Propagation

Since the egocircle collapses the 3D information down to 2D information (angle and range), the data format of the egocircle measurement module is compatible with a laser scan. The laser scan information populates the egocircle data structure, whose contents get propagated and updated as the robot maneuvers. Similar to a laser scanner, the egocircle evenly divides the angular space into n_{circ} cells or buckets, with each containing a list of 2D points that fall within the cell's angular range. Generating an egocircular map from the stored data entails performing a min operation over all egocircle buckets individually. For shorthand, we write

$$L_m(r_{cir}) = \min(\rho(\mathcal{L}_{ego}(r_{cir}))), \tag{9}$$

where r_{cir} is the coordinate indexing into the egocircle structure, and $\mathcal{L}_{ego}$ is the collection of buckets. The process above renders a 1D measurement "image" L_m from all points stored in memory. It is equivalent to a 360 degree laser scan sensor reading whose angular resolution is $n_{circ}/(2\pi)$.

Following §5.2, propagation of the stored points involves transforming them under the motion induced pose change $g_{move} \in SE(2)$ from one time point to the next, where g_{move} gives the coordinate frame of the old robot pose relative to the new robot pose. Define the egocircular coordinate vector $p_{cir} = (\rho, \theta)^T$ and the Cartesian coordinate vector $p = (x, y)^T$. Consider the mapping from egocircular coordinates to planar Cartesian coordinates T_{l2p} and vice-versa T_{p2l},

$$p = T_{l2p}(p_{cir}) = \begin{bmatrix} \rho\cos(\theta) \\ \rho\sin(\theta) \end{bmatrix} \quad \text{and} \quad p_{circ} = T_{p2l}(p) = \begin{bmatrix} \sqrt{x^2+y^2} \\ \mathrm{Arg}(x+jy) \end{bmatrix}, \tag{10}$$

both with reference to the viewer/camera frame modeled as an $SE(2)$ frame. The new egocircular coordinates p'_{cir} of a stored point p_{cir} are:

$$p'_{cir} = T_{l2p} \circ g_{move} \circ T_{p2l}(p_{cir}). \tag{11}$$

Identifying the new bucket that the point should be moved to involves partitioning the angular values according to the egocircle's angular resolution. To speed up calculations, our implementation stores points using Cartesian coordinates and keeps track of the minimum range per cell. Points are removed once they lie outside the radius ρ_{max} associated with the local egocircular map. When incorporating new measurements from the most recent depth image, range-based clearing occurs with the stored egocircle data.

This design allows the egocircle to track multiple points in each direction and to quickly return the distances to the nearest obstacle in each direction. With propagated and depth image updated egocircle data, synthesis of the 1D range image per Eq.9 contains historical knowledge regarding the local environment. Figure 11 depicts a navigation scenario for a simulated Turtlebot mobile robot. The robot travels upwards relative to the overhead views in the bottom row, turns right, then proceeds rightwards. Visualizations of egocircle generated measurements are in the top row (but with a coordinate system orientation roughly matching that of the world). The red lines delineate the FOV of the mobile robot. Historical data sensed by the robot is contained in the egocircle measurements outside of the FOV. Note that, in the rightmost egocircle measurement, a section of the upper surface of the wall (just below and to the left of the robot) does not exist in the egocircle data structure. As the robot turns towards the right, FOV limitations mean that this small portion of the wall never gets sensed, hence the missing data. By maintaining a local, approximate cost map in planar space, the egocircle provides a means to rapidly score candidate robot trajectories. The following subsections describe the different cost functions evaluated and contributing to the total score of a candidate trajectory.

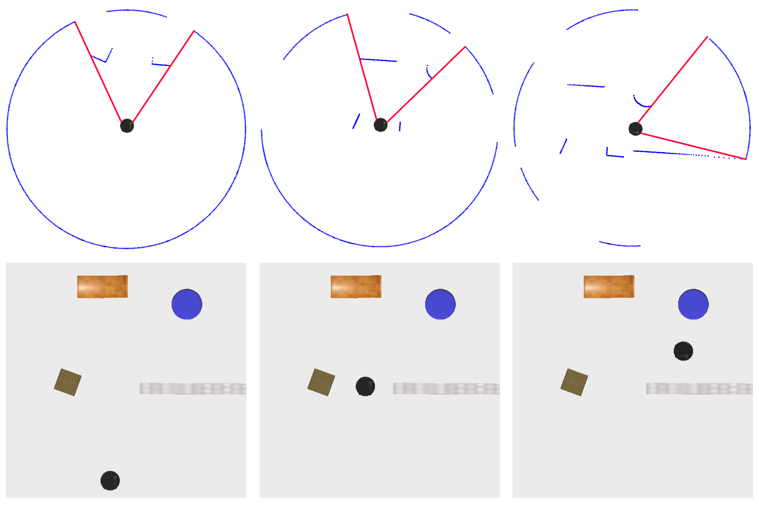


Fig. 11 Visualizations of the egocircle measurement predictions based on the stored data. The top row depicts the egocircle measurements where only those points that would be visible to a laser scanner are plotted (occluded points are not). The origin corresponds with the camera origin, but the orientation of the egocircle measurement has been adjusted to roughly align with the global overhead views of the bottom row. The red lines are FOV limits.

5.3.1 Egocircle Trajectory-Based Cost Functions

The purpose of the egocircle representation is to replace the grid-based cost functions utilized by DLP and many other non-perception space methods. Figure 12 depicts a navigation scenario with the global trajectory (in color green), the actual navigated trajectory (in color red), and a candidate future trajectory (in color yellow). This candidate trajectory should be scored in order to identify an appropriate fitness relative to the constraints of following the established trajectory and avoiding collisions. The local egocircle map contains only the world sensed information within the depicted radius. It generates a local egocircular range scan from the information (color blue),

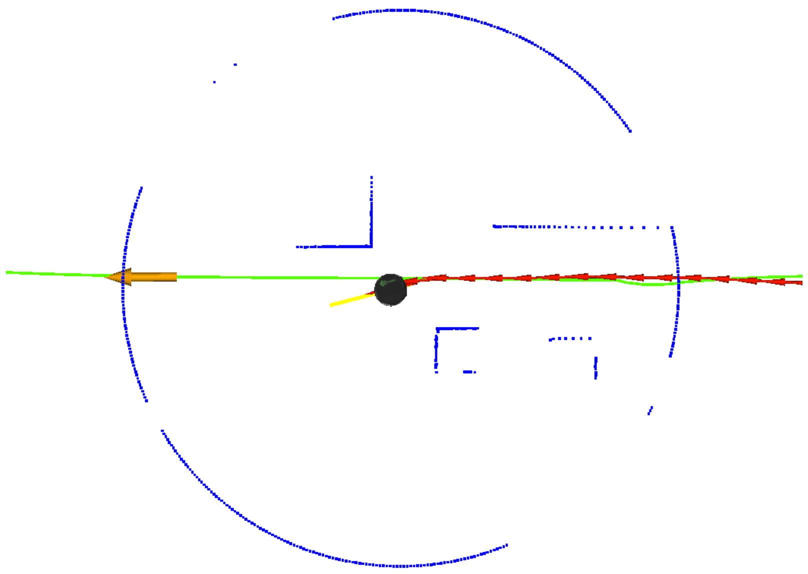
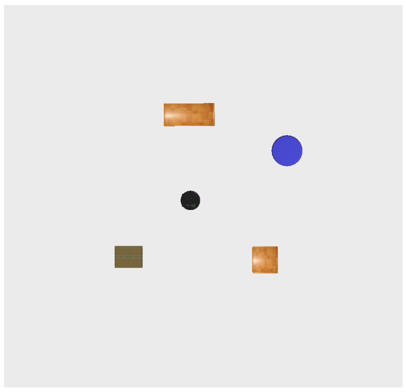


Fig. 12 Local egocircular representation with global path (green), sampled local path (yellow), local goal (orange arrow) and odometry (red arrows). The robot moves leftwards.

which is used to calculate some of the trajectory costs. The particular costs needed for a functional Move Base implementation include the obstacle cost function, the go-to-goal cost function, and the path comparison cost function. These costs were depicted earlier in Fig.5 based on a local occupancy grid. A description of these costs and the analogous egocircle implementation representation will be given below.

The trajectory scoring is meant to provide a rank ordering of the sampled trajectories from best to worst for prioritizing collision checking and trajectory safety assessment. It does not need to involve a precise representation of the robot model nor scoring functions, as long as the model is approximately correct and the scoring functions are monotonically correct over large swaths of the local area around the mobile robot. Therefore several simplifications are made to improve computational runtime.

An important simplification is using an inflated egocircle range scan in order to treat the robot as a point for certain tasks. Conceptually, a circle of radius r_{ins} is placed at the location of each point represented by L_m and a new egocircle scan is generated based on the ranges to these inflated points (see Fig.13). The inscribed radius r_{ins} of the robot is used to ensure that the result is liberal. This requires first finding the subset of points in the egocircular map within a specified distance d_s of a given pose $p_s = (\rho_s, \theta_s)^T$. We approximate this as:


(a) Simulated Environment

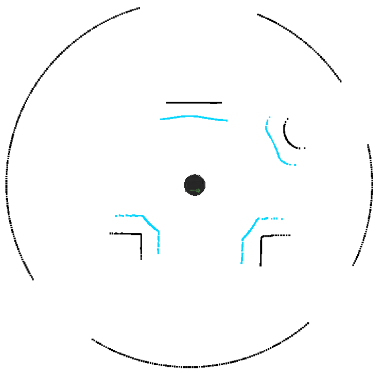

(b) Original & Inflated Egocircles

Fig. 13 Overhead views showing the simulated environment (left) and the egocircle representation (right). The original egocircle is shown in black while the inflated egocircle is shown in cyan.

$$\beta(p_s, d_s) = \left\{ p_b \in L_m \,\middle|\, (\theta_s - \theta_d) \leqslant \theta_b \leqslant (\theta_s + \theta_d) \right\} \tag{12}$$

where $\theta_d = d_s / \rho_s$.

Let $T_{m2p}(L, i)$ map the ith element of egocircle range scan L to egocircular coordinates. The inflated egocircle can then be expressed as follows:

$$\begin{aligned} L_{infl}(i) &= \min_{j \in B(i)} L_m(j) - r_{ins}, \text{ where} \\ B(i) &= \left\{ j \in n_{circ} \,\middle|\, T_{m2p}(L_m, i) \in \beta(T_{m2p}(L_m, j), r_{ins}) \right\} \end{aligned} \tag{13}$$

Obstacle Cost Function

The obstacle cost function reflects the cost of traveling close to obstacles. The obstacle cost of a trajectory is a function of the obstacle costs of the poses in the trajectory. The obstacle cost of query point p is a function of the distance to the nearest obstacle point, represented as $d_{min}(p)$. The obstacle cost c_{obs} is

$$\begin{aligned} c_{obs}(p) &= c_{obs} \circ d_{min}(p) \\ &= \begin{cases} -1, & \text{if } d < r_{ins} \\ \bar{c}_{obs} \exp^{-w(d - r_{ins})}, & \text{if } r_{ins} \leq d < r_{max} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{14}$$

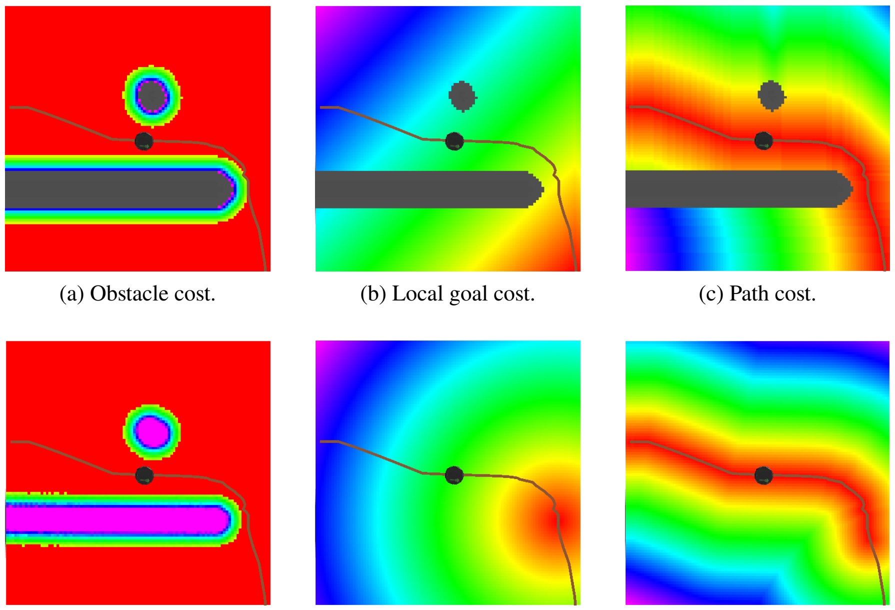
where $d = d_{min}(p)$, and $\bar{c}_{obs}$ is a predetermined constant cost. The values r_{ins} and r_{max} represent the nearest permissible distance and the distance beyond which an obstacle has no cost. An obstacle cost of -1 means that a pose definitely collides. Depending on the geometry of the robot (i.e., if it is elongated or otherwise not a circle), it is possible for colliding poses to receive nonfatal obstacle costs. If any pose in a trajectory collides, the trajectory is assigned the fatal cost of -1. Otherwise, the trajectory is assigned the obstacle cost of the last pose in the trajectory.

The standard costmap-based obstacle cost function uses a distance map to implement d_{min}. The distance map is computed such that each cell contains the Euclidean distance to the nearest occupied cell in the occupancy grid. The egocircular representation does not admit such a calculation since it is a boundary based polar model of 3D space (it does not measure space according to discretized area or volume). Instead, d_{min} is brute force computed between the query pose and a local subset of the egocircle as follows:

$$d_{min}(p) = \min_{b \in B} \text{dist}_p(p, b)) \tag{15}$$

where $B = \beta(p, r_{max})$ and dist_p returns the distance between polar points p and b using the law of cosines.

To understand this distance based cost function, it is best to examine the first column of costmap images in Fig.14. The red regions correspond to 0 values as those locations are far from the obstacles, which are the black regions in the image. The coloring trend goes to blue/purple as the distance from the query location to the nearest obstacle point lowers to be within the interval $[r_{ins}, r_{max}]$. The black obstacle regions would give -1 values. These should be very large or infinite costs, however the Move Base implementation checks for–and rejects trajectories with–negative scores. The top image, Fig.14a, is the grid-based costmap while the bottom image, Fig.14d, is the egocircle costmap evaluated using Eq.14 over the same grid as the costmap, where each point in the 2D grid is converted to polar representation. The two functions have similar scores outside of the occupied regions. The pairwise computations for the egocircle implementation, per pose tested, are quadratic but based on two low cardinality point sets. The grid-based distance computing scheme is linear in the local occupancy grid area [101], thus it too is quadratic in time cost but with larger base values. Once computed, grid-based trajectory costs are constant per robot pose tested. As with collision checking, the data preparation time cost is near zero for the perception space approach but has a high slope in per pose scoring time. The world centric model has a non-negligible time cost to build the scoring maps and a small constant cost to score per pose. The perception space method will be faster than the grid-based approach until a large enough quantity of poses is sampled. The number of poses to score grows linearly with the number of trajectories to score.


(a) Obstacle cost. (b) Local goal cost. (c) Path cost.

(d) Obstacle cost. (e) Local goal cost. (f) Path cost.

Fig. 14 Visualization of the trajectory scoring as cost maps computed on an occupancy grid (top row) and from egocircle (bottow row). The grid points are converted to polar representation and scored according to the described scoring functions. Low cost is red and high cost is blue/purple. Points lethally close to obstacles are black in the occupancy grid and purple in the egocircle. The brown curve is the global path to follow.

Goal Point Cost Function

The purpose of this cost is to reward candidate trajectories whose terminal points are close to the local goal point. In a grid-based scoring strategy, the local goal is selected as the first pose on the global path that exits the local costmap. The distance map for the go-to-goal cost warps around obstacles and reflects the true cost-to-go if the occupancy map is correct (see Fig.14b). Under an egocircle representation, two deficiencies occur: (i) the true cost-to-go requires more computation when there is an obstacle between the evaluation point $p_{cir,i}$ and the local goal p^*_{loc}, and (ii) the true cost-to-go cannot be ascertained when there is an obstacle between the robot's camera (located at the origin) and the local goal. For the first case, we simply compute the distance as though there were no obstacles. We avoid the second case by selecting local goal as the last unobstructed pose on the global path that lies within the egocircle. Poses are classified as obstructed or unobstructed by using the inflated egocircle $r_{inflated}$.

Let the index set J be defined as follows:

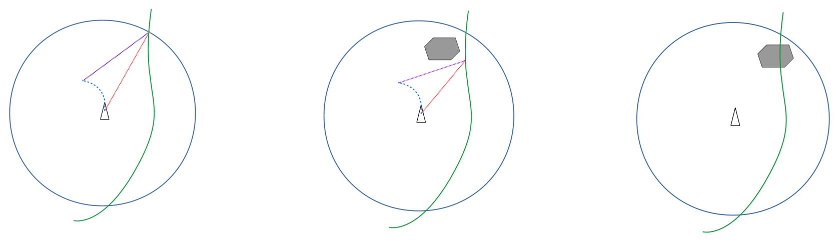

(a) No obstacle (b) Occluding Obstacle (c) Global Path Collision.

Fig. 15 Selection of local goal for goal point cost function without (a) and with (b) an occluding obstacle. The right-most plot (c) depicts a scenario that would trigger global replanning.

$$J = \left\{ j \in \mathbb{N} \;\middle|\; g^*_j = g^*(t_j) \text{ for } t_j \in \mathbb{R}^+ \right\} \tag{16}$$

where $g^*(t)$ is the global path and t_j indexes into it to create a set of global path waypoints.

Pose p_{loc} is unobstructed if $\rho_{loc} < r_{inflated}(Angle(p_{loc}))$. Poses that are less than $2 * r_{ins}$ behind an inflated point are definitely in collision, but poses further behind may simple be occluded. Consequently, we classify each pose as either safe, colliding, or occluded. If $g^* t_j$ is safe and $g^* t_{j+1}$ is colliding, the global path leads to collision and replanning is triggered. Otherwise, the local goal is selected as follows:

$$\begin{aligned} p^*_{loc} = g^*(t_j) \max_{j \in J} j \;\Big|\; & \text{dist}(g_{robot}, g^*(t_j)) < \rho_{circ} \\ \text{and} \quad & L_m \circ L_{infl}\left(\text{Angle}(g^{-1}_{robot} g^*(t_j))\right) > \text{dist}(g_{robot}, g^*(t_j)) \end{aligned} \tag{17}$$

where g* is the global path and g_{robot} is the $SE(2)$ pose of the robot in the world frame.

Figure 15 summarizes the different cases determining the selection of the local goal. If there are no occluding obstacles, the local goal is the last pose on the global path that lies within the egocircle radius (Fig.15a). Otherwise, the local goal is the last unobstructed pose on the global path (Fig.15b). However, if the global path definitely collides (Fig.15c), local planning is aborted and global replanning occurs.

This cost function is adapted to the star-like free space region associated to the current robot pose and based on its local environment, as captured by L_m. It employs the line-of-sight visibility property, from the robot pose (the origin of the egocircle) to the goal point, to establish the scoring.

Visualization of the go-to-goal cost function is given in the second column of Fig.14 for the grid-based method (top) and the proposed heuristic (bottom). The grid point values would be the scores associated to the point $p_{cir,i}$ if it were located at those grid points. Note that the location of the goal point p^* differs between the two methods. The important characteristic is that the proposed heuristic approximately matches the monotonic increase in the go-to-goal cost of the equivalent occupancy grid cost (see Fig.14b versus Fig.14e). The difference in the shape of the cost level sets is due to the grid-based method using Manhattan distance rather than Euclidean distance.

Path Cost Function

In addition to the go-to-goal cost, there is another cost associated to the global path. This one scores the sample trajectory's terminal point against the global path to identify how close it is to a point on said path. Relying again on the star-like properties of the free-space region described by the polar egocircle estimate L_m, any point on the global path that is obscured by a point on the inflated egocircle is not considered as reachable from the terminal point of the candidate trajectory. With the index set J defined as in Eq.16, the local set of visible points on the global path is the subset:

$$J_{local} = \Big\{ j \in J \;\Big|\; \text{dist}(g_{robot}, g^*(t_j)) < \rho_{circ} \\ \text{and} \quad L_m \circ r_{inflated}\left(\text{Angle}(g_{robot}^{-1} g^*(t_j))\right) > \text{dist}(g_{robot}, g^*(t_j)) \Big\} \tag{18}$$

where g_{robot} is the $SE(2)$ pose of the robot in the world frame. The set consists of indices to unobstructed global path poses within the egocircle domain. Pose obstruction is evaluated as in the goal cost function. The path cost is then

$$C_{path}(p, J_{local}) = \min_{j \in J_{local}} \text{dist}(p, \text{Pos}(g_{robot}^{-1} g^*(t_j))), \tag{19}$$

where $\text{Pos}(g)$ grabs the translational coordinates of the $SE(2)$ element g. Again, the distance calculation in the above equation uses the law of cosines to generate the distance using polar representation for point positions. The cost is basically the straight path length from the trajectory terminal point given by p to the path waypoint in the robot's frame given by $\text{Pos}(g_{robot}^{-1} g^*(t_j))$. Global path waypoints obscured by an obstacle do not factor into the cost. The third column of Fig.14 depicts the traditional grid-based path cost (top) and the heuristic polar path cost (bottom). For the latter, regions occluded by obstacles and lacking line of sight have a cost based on the nearest visible path point. As with the previous cost comparison, the overall trend matches between the two implementations in the free-space regions, with the level sets once again having different shapes due to the different distance metrics utilized.

5.4 Working with Stereo Cameras

With additional processing, stereoscopic camera configurations provide similar structural information as depth or range sensors (as can multi-camera setups involving more than two cameras with overlapping fields of view). The process for estimating depth, triangulation, requires matching pixels on the left camera to those of the right camera (the two pixels should represent the same world point, usually on an object surface). A popular stereo configuration for robotic systems offsets two cameras horizontally to provide image pairs similar to human binocular vision (see Fig.16).

In the parallel stereo configuration, epipolar lines are horizontal lines in the image plane (e.g., dash dotted line in the figure). For each pixel in the left image, the corresponding pixel in the right image is on the epipolar line. Limiting the search to a vicinity of the epipolar line constrains the pixel matching search space for more efficient stereo matching. Image rectification can be applied first to transform them into one common image plane so that all epipolar lines are horizontal. When rectified, matching pixels are displaced horizontally between the left and right rectified images. The disparity measures the pixel horizontal distance, usually in the r^1 coordinate, between two corresponding points in the left (L) and right (R) images, $\delta = r_L^1 - r_R^1$. The depth value z is a function of the disparity and camera model parameters.

$$z = \frac{b\,f^1}{\delta} \tag{20}$$

where b is the baseline representing the distance between two camera centers, f^1 is the focal length of the horizontal coordinate, and δ is the disparity.

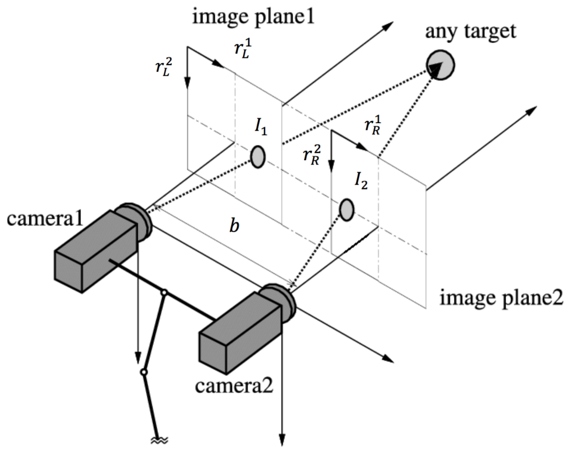


Fig. 16 Stereo camera model [106]. (r_L^1, r_L^2) and (r_R^1, r_R^2) are coordinate systems of two image planes. b is the baseline of stereo camera model. I_1 and I_2 are pixel values of the same target point in the left and right image planes.

Following the taxonomy of stereo matching algorithms [107], approaches divide into local and global methods. The traditional block matching algorithm is one of the local methods. This algorithm first extracts a small patch around the pixel in the left image, and horizontally shifts the patch along the epipolar line by candidate distance values (in a predefined disparity range) in the right image. The estimated disparity is the one that minimizes the difference between the patch in the left image and the shifted patch region in the right image. This difference can be represented by cost function. One traditional cost is the sum of squared differences (SSD).

$$\sum_{(\zeta^1,\zeta^2)\in\mathcal{N}(r^1,r^2)} (I_1(\zeta^1,\zeta^2) - I_2(\zeta^1+\delta,\zeta^2))^2 \tag{21}$$

where $I_1(\zeta^1,\zeta^2)$ is a pixel value in the left image and $I_2(\zeta^1+\delta,\zeta^2)$ is a pixel value in the right image. Several alternative cost functions exist for the matching optimization, such as normalized cross-correlation (NCC), sum of absolute differences (SAD), mean of absolute differences (MAD), etc. [107]. [108] also introduces the semi-global block matching algorithm that integrates local pixelwise matching and global smoothness constraints. This approach has better performance when dealing with varied illuminations, occlusions and low texture surfaces. Moreover, in order to reduce computational complexity and obtain smoother disparity, some optimization and refinement techniques are integrated [107]. Real-time disparity estimation implementations exist based on specialized hardware approaches, such as via onboard FPGA [109, 110] or graphical processing units [111]. Once the disparity image is estimated, generation of the depth image using Eq.20 is immediate and can be used within the PiPS local planner. Though [105] employs disparity space for navigation, we advocate its conversion to depth space due to the fact that calculation of the equivalent Cartesian point representation is more efficient for propagation of the points in the egocylinderical and egocircular representations (and easier to write as a set of operations).

6 Benchmarking Navigation Methods

To aid in the evaluation and comparison of navigation frameworks and strategies, this section describes a set of ROS/Gazebo based environments and associated initial/terminal point synthesis methods for generating repeatable navigation scenarios. While it is preferable to deploy in real-world scenarios, doing so is more difficult due to the need for the other components of a robotic navigation system to be working perfectly, the setup and real-estate costs of creating and maintaining the environment, the lack of configuration flexibility [112]. Plus more universal evaluation by other researchers in these actual environments would be difficult. The value of ROS/Gazebo is that highly repeatable experiments are possible in a diverse array of worlds. The same experiments can be performed by anyone with access to a system configured with ROS/Gazebo and to our publicly available benchmark worlds and testing configurations [113]. Furthermore, in our experience developing the original local PiPS algorithm [100], we found little difference in performance between deploying in a well designed Gazebo world and in our actual office environment.

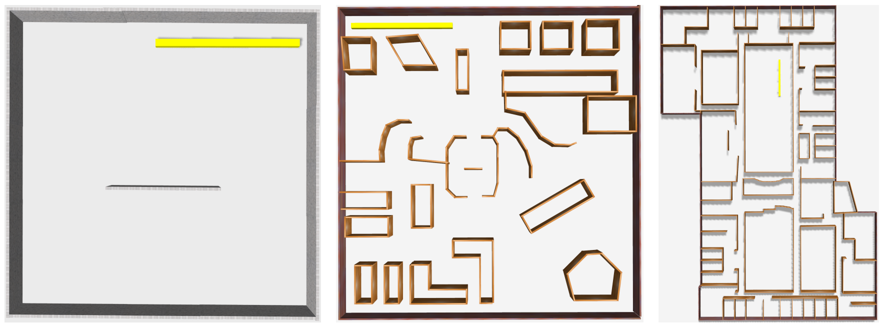
The testing protocol for the benchmark scenarios includes Monte Carlo runs employing multiple point-to-point navigation instances that generate statistical outcome models for comparison purposes. Important metrics include completion rate, path length, and travel time. Though Gazebo simulations are not perfectly deterministic, the outcomes should be close enough that the final Monte Carlo statistics will have low variation (i.e., inter-experiment variance is low).

6.1 World Synthesis

Because navigation is a generic capability expected of mobile robots, the environments where robots may be deployed will vary in scale, structure, and obstacle density. The proposed benchmark consists of several synthetic worlds modeled after environments observed to exist here on our university campus. These synthetic worlds are called *sector world*, *campus world*, and *office world*.

1. **Sector World.** (Fig.17a) The sector world consists of a single large room, partially divided in the middle by a wall running from left to right. It is intended to represent locations that are essentially large open areas dotted with a wide assortment of obstacles. The lack of known obstacles means that the global planner will generate simple piecewise linear paths from a start pose to a target goal pose.
2. **Campus World.** (Fig.17b) The campus world is intended to model the outdoor free space of a university campus, where adjacent structures and landscaping force one to follow specific paths, and open areas admit more free form movement across them. Consequently, it consists of several relatively large open areas connected by narrower corridors. The narrow corridors are well defined and generally clear of miscellaneous obstacles. The open areas (e.g., open quads) are presumed to have multiple purposes and therefore will contain a randomized assortment of obstacles. When considering particular global planning or world modeling strategies, some areas of this world are best simplified through a graph-based or topological model of the space (e.g., corridors linking open spaces), whereas other areas benefit from spatial world models.
3. **Office World.** (Fig.17c) The office world is based on digital architectural floor plans of the fourth-floor of the building containing our lab, provided by the interior designers of the building after it was remodeled. It serves as an example of a real office indoor scenario, with long hallways connecting open cubicle areas, enclosed offices, and larger conference rooms and laboratory spaces. While halls are narrow enough to be easily blocked, there are generally alternate routes available. Doors that are usually closed have been replaced with walls.

For each of these worlds, there is an associated global map containing only the permanent structural elements (the walls). Global paths generated from the map



(a) Sector World. (b) Campus World. (c) Office World.

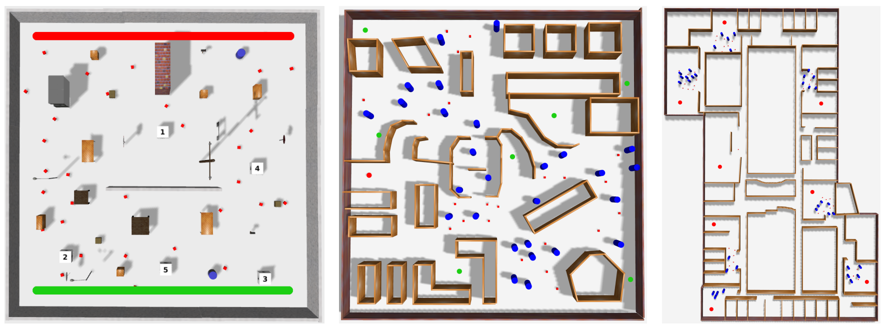
Fig. 17 Overhead views of the benchmark worlds. The 10m yellow bar provides a means to assess the scaling of the three worlds. The office world is rotated.

will typically be infeasible due to unmodeled obstacles. Some paths may be blocked for the narrower passages, should there be objects (randomly) placed within them. The next section describes how specific navigation scenarios are configured and instantiated.

6.2 Scenario Configuration

Monte Carlo runs provide statistical insight on the performance outcomes from a large sample of data. In each Monte Carlo run, one or more characteristics of the experiment are determined by a provided random seed. The seed ensures that experimental conditions are variable and repeatable. An important aspect of a scenario configuration is the type and placement of obstacles. We distinguish between two general categories of obstacles: laser safe and laser unsafe. Laser safe obstacles have vertical sides or an invariant occupancy profile from the floor up to the height of the robot. This property ensures constant cross-sectional geometry and satisfies the assumptions of laser-scan based planning approaches regarding world geometry for successful collision avoidance. Laser scan approaches should be able to safely avoid laser safe obstacles but may not be able to avoid laser unsafe obstacles. Figure 19 depicts some of the safe and unsafe obstacles available for the scenarios, as available through the Gazebo model database. The specific randomized configurations for each world are as follows:

1. **Sector World.** (Fig.18a) Starting poses are sampled from a line (depicted in red) running inside and parallel to the north wall of the world and place the robot facing inwards. The coordinates for this region are $x \in [-9, 9]$ and $y = -9$. Goal poses are sampled from a parallel line just inside the south wall (depicted as the green line), whose coordinates are $x \in [-0, 9]$ and $y = 9$. All navigation tasks require traveling from one side of the world to the other. The area between the start



(a) Sector World. (b) Campus World. (c) Office World.

Fig. 18 Worlds annotated with start (red) and goal (green) points or regions. For sector world, the start and final points are selected from regions. The campus world has a single start point (larger red circle) and multiple goal points (green circles). The office world start and goal points are randomly chosen from the red circles in the map. The worlds also show examples of random positions populated with obstacles. The blue objects are randomly placed laser safe obstacles. The smaller red dots are laser unsafe obstacles.

and goal lines is populated by laser safe obstacles at fixed locations in the world. Low and medium density configurations exist. Manual placement of the obstacles was done with the aim of creating an approximately uniform distribution with moderate clearance between obstacles. If desired, laser unsafe obstacles can also be randomly placed throughout the same area of the environment.

2. **Campus World.** (Fig.18b) For the campus world, there is one start pose (the red dot) and seven candidate goal poses. A given scenario will randomly select one of these predefined goal poses. A specified number of obstacles are uniformly distributed among the primary open areas of the world (random position and orientation). The obstacle type is randomly chosen as either a blue cylinder (laser safe) or a small red box (laser unsafe). As obstacles are placed, a minimum obstacle-spacing threshold ensures that the navigation task remains feasible.
3. **Office World.** (Fig.18c) In the office world, there is a fixed set of locations (the red points). Start and goal poses are randomly selected from a list of locations around office. In order to reduce the duration of experiments in this significantly larger world, the goal pose is randomly selected from only the three poses nearest to the start pose. Obstacles are randomly placed in designated regions (free space between start/goal poses and corriders) using the same approach as in Campus World.

6.3 Benchmarking

Benchmarking a particular navigation strategy involves multiple runs with a predetermined set of random seeds. Comparison with another navigation method re-

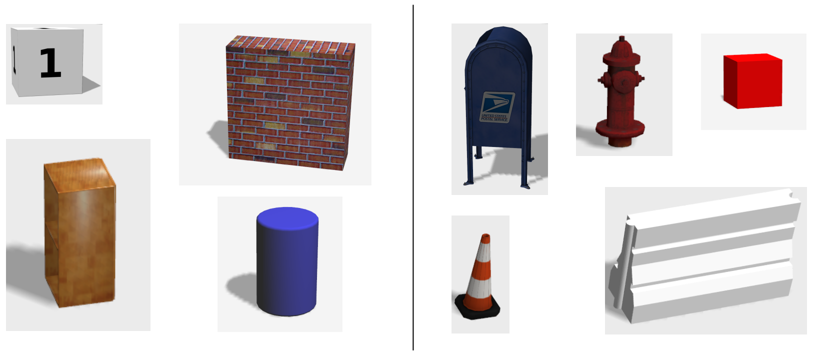


Fig. 19 The left area depicts laser safe obstacles while the right area depicts laser unsafe obstacles. The top-right obstacle is a custom box with very short height that cannot be detected by the laser scan.

quires using the same random seed set. An experiment instance proceeds through four stages:

1. World Setup: The test Gazebo world is loaded and a given robot model is placed within in it.
2. Task Setup: The robot is moved to the starting pose and all obstacles are added to the environment. The specified robot navigation method is then initialized.
3. Navigation: The goal pose is sent to the robot controller and a timer is started. The experiment remains in this stage until one of the following conditions is met:

   a. *Succeed:* the robot reaches the goal without collisions;
   b. *Bumper Collision:* the robot bumps into an obstacle en route to the goal;
   c. *Aborted:* the controller reports that planning has failed;
   d. *Timed out:* the timer reaches 10 minutes.

   During execution of the navigation scenario, pertinent scoring metrics are maintained or accumulated as needed.
4. End: The value of the timer is saved as ’path time’. The final path length is also saved, along with the condition that ended Navigation. The controller is shutdown.

To compare performance, several metrics are calculated and stored for each experimental instance. Popular metrics for evaluating robot navigation frameworks include success rate, defined as the number of successful runs divided by the total number of runs; path distance, defined as the robot path length travelled from start to goal; and path time, defined as the time required for robot to reach the goal. The latter two statistics are computed from the subset of successful runs. These metrics measure the performance of planners from different perspective including robustness, efficiency, and optimality. A good planner should perform well in these metrics. Potential navigation parameters to consider or configure include the replanning rate, the recovery behaviors, and the local map radius.

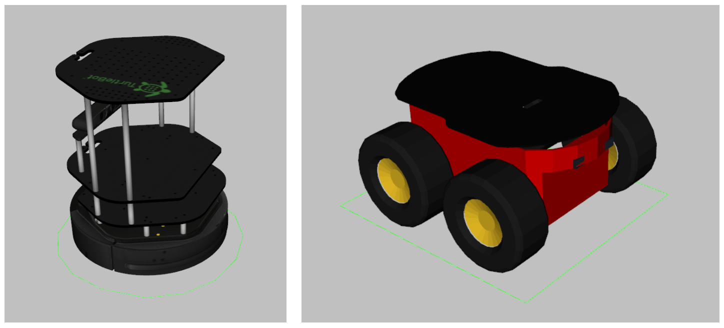

(a) Turtlebot (b) Pioneer

Fig. 20 Robot models used in the experiments. Green boundaries are footprints of robots.

7 Navigation Experiments

Evaluation of the described perception space approach to navigation (PiPS DLP) will consist of Monte Carlo testing on the benchmark navigation scenarios. Comparison will be made with the standard Move Base implementation (baseline DLP), designed for use with a laser scanner. The first scenario will test in the sector world, with both laser-scan friendly and laser-scan unfriendly sector world instances. No laser unsafe obstacles are added to friendly instances, while 30 are added to unfriendly instances. The purpose of the experiment is to show that the perception-space approach performance matches the classical methods under environmental conditions appropriate for laser scanners and outperforms the classical methods under more general conditions. The second set of scenarios will test the perception space and classical navigation algorithms on the other two benchmark worlds, campus world and office world.

An additional experimental variable will be the mobile robot type, where the robot geometry will vary. The two robots will be the Turtlebot, Fig.20a, and the Pioneer mobile robot, Fig.20b. The Turtlebot is a two-wheel differential drive robot platform with cylindrical robot configuration and a circular base. For collision checking purposes, it is modeled as a cylinder with 0.2m radius. Depth images on the Turtlebot are captured by a Kinect camera. The Pioneer is a four-wheel skid-steer drive robot platform with a non-circular base, and non-cylindrical configuration. The robot is modeled as a 0.56m x 0.5m rectangular box and configured to have a Realsense R200 depth camera. Since we are only evaluating the ability of the controllers to handle the changed geometry, the Pioneer is simulated as a Turtlebot base with the Pioneer's chassis geometry on top. Since neither of these robots is equipped with a laser scanner, we use the Depth Image to Laser Scan ROS package [114] to create virtual scans based on the 10 rows of pixels nearest the optical center of the depth camera. Each scenario involves 50 Monte Carlo experimental runs for each local planner and each robot model.

Global replanning and recovery behaviors (see §5) are both enabled in all experiments. The time threshold of recovery behavior is defined as $controller\_patience$.

Table 4 Common Parameters

Goal Position Tolerance	Goal Orientation Tolerance	Global Replanning Frequency	Controller Patience
1m	2π	1	3

Planner Patience	Local Planning Frequency	$\#v$ Samples	$\#\omega$ Samples
5	5	6	20

The purpose of the recovery behavior for baseline DLP is to clear space in the local costmap. Since PiPS DLP does not use a local costmap, its recovery behavior is altered to rotate the robot in 90° increments in an attempt to point the robot away from whatever was obstructing it. Recovery behaviors are also used if $planner\_patience$ elapses without a valid global plan being found. The size of the local region considered during local planning is also an important parameter. A 5mx5m square costmap with 5cm resolution is used in baseline DLP, while an egocircle with 512 cells and a radius of 3m is used in PiPS DLP. Common parameter values are given in Table 4. The forward sim time values are different: 1s for baseline DLP, 2s for PiPS DLP. Though PiPS DLP can be run with a local planning frequency of nearly 30Hz, here we use the default frequency value of baseline DLP (5Hz) for all experiments. The global costmaps of both approaches are populated using the virtual laser scans described above. The global costmap of PiPS DLP is also populated with the locations of collisions detected by the collision checker. The values for all customized parameters are available in the configuration files at [113].

7.1 Sector World with Laser Safe and Unsafe Obstacles

In the medium density sector world with laser safe obstacles, both baseline laser scanner DLP and Ego-Centric perception-space planning have nearly 100% success rates (see Table 5), which shows that our approach has similar performance under normal environments. The failure case abbreviations are bumper collision (BC), aborted (AB), and time-out (TO). The Pioneer and Turtlebot robots each only had one AB out of 50 runs, with the rest being successful. The path lengths taken by all of the robots were within 2% of each other, indicating that they all found comparable paths. The completion times of the PiPS approach were close to but a few seconds more than the laser scan baseline.

After randomly adding 30 laser unsafe obstacles with 1m and 1.2m minimum distance between each other for the Turtlebot and Pioneer robots, respectively, the success rate of the baseline laser scanner DLP drops to 40% and 24%. The egocentric PiPS approach still has good performance with a success rate of 94% and 84%, see Table 6. Again, the average path lengths and completion times of both methods are similar, except for PiPS approach with Pioneer robot. The additional

Table 5 Results for sector world with laser safe obstacles.

Sector world with laser safe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	100%	40.98s	19.62m	0% / 0% / 0%
PiPS DLP	98%	42.96s	19.47m	0% / 2% / 0%
Pioneer				
Baseline DLP	100%	43.42s	20.03m	0% / 0% / 0%
PiPS DLP	98%	46.36s	19.74m	0% / 2% / 0%

Table 6 Results for sector world with laser safe and randomly placed laser unsafe obstacles.

Sector world with laser unsafe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	40%	43.62s	19.68m	58% / 2% / 0%
PiPS DLP	94%	46.37s	19.68m	0% / 6% / 0%
Pioneer				
Baseline DLP	24%	43.45s	19.76m	76% / 0% / 0%
PiPS DLP	84%	56.64s	20.51m	8% / 8% / 0%

maneuvers necessary for keeping clear of laser unsafe obstacles led to reduced forward speeds for PiPS, yielding completion times 30% larger than the baseline. The success rate of Pioneer PiPS DLP is lower than that of Turtlebot. Most of the additional failure cases are caused by bumper collision. Due to the geometry of Pioneer and the programming of the recovery behavior, it can collide with obstacles while executing the recovery behavior.

7.2 Campus World and Office World

In campus world and office world, 50 obstacles are randomly selected from laser safe and unsafe obstacles with the same minimum distance offsets as in sector world. In all cases, the PiPS modification has improved success rates versus the equivalent baseline implementation. Furthermore, the success rate of the baseline navigation scheme decreases when switching from the cylindrical robot to the rectangular box robot while the success rate of PiPS DLP does not. PiPS completion times continue to be a few seconds longer, as seen in the Sector world cases, while the path lengths remain similar to those of the baseline cases. Having roughly comparable path lengths indicates that the scoring system of the egocircle representation is capable of providing a ranked ordering of the sampled trajectories similar to that of the baseline local planner. In campus world, the completion rates of Pioneer PiPS DLP are larger than those of Turtlebot. One reason could be the different minimum distances between obstacles (1.2m for Pioneer, 1m for Turtlebot). Though the distances are designed to give similar clearance to both robots, the greater spacing may disproportionately

Table 7 Results of campus world with randomly placed laser safe and unsafe obstacles.

Campus world with randomly selected laser safe and unsafe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	68%	38.82s	18.44m	30% / 2% / 0%
PiPS DLP	80%	50.03s	20.37m	0% / 20% / 0%
Pioneer				
Baseline DLP	50%	40.17s	19.26m	48% / 2% / 0%
PiPS DLP	88%	49.27s	21.43m	8% / 4% / 0%

Table 8 Experiment results of office world with randomly placed laser safe and unsafe obstacles.

Office world with randomly selected laser safe and unsafe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	72%	100.93s	51.03m	28% / 0% / 0%
PiPS DLP	92%	103.54s	48.70m	0% / 8% / 0%
Pioneer				
Baseline DLP	66%	98.18s	64.25m	34% / 0% / 0%
PiPS DLP	96%	104.96s	61.21m	2% / 2% / 0%

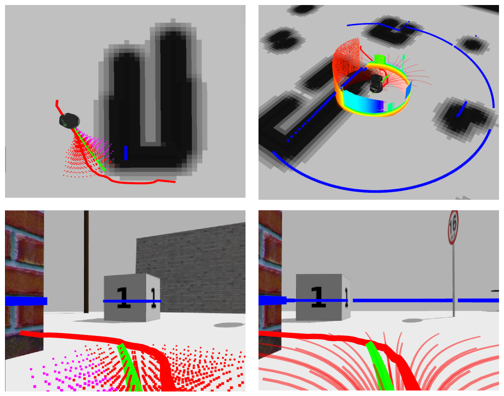


Fig. 21 Visualizations of the navigation process of baseline DLP (left) and perception-space DLP (right). The top row has visualizations in the 3D world space from an external reference frame. The bottom rows has visualizations of the same information overlayed on the robot's camera view.

reduce the problem of local minima for the Pioneer (see §7.3). It should also be noted that the majority of failures are due to aborted navigation as opposed to collisions. The PiPS method is successful at avoiding collisions. The abort outcomes reveal failure modes of the hierarchical planner.

7.3 Review of Outcomes

Here, we look a little more closely at the two different hierarchical navigation implementations and also review the causes of failure. First, we review how the baseline and PiPS navigation methods compare. The navigation information associated to the two implementations is shown in Fig.21. The top row provides an overhead view of the mobile robot during navigation past and around a wall, with the local segment of the global path also plotted. Due to the expansion of the occupancy grid by the robot radius, the occupied regions are slightly thicker than the true occupancy. The blue curve segments on the plots are from the sensor information (laser scan or egocircle) and indicate the true locations of the obstacle surfaces. The depicted occupancy grids communicate the information that the baseline DLP system would have during planning, while the red and magenta points are sampled and scored robot poses. The magenta poses correspond to trajectories with collisions, while the red poses correspond to safe trajectories. The thick green curve is the chosen local path to follow. It is the trajectory with the lowest weighted sum of obstacle cost, local go-to-goal cost, and path cost. In the case of the PiPS approach, it was also deemed to be collision free based on the egocylindrical model. The baseline DLP system is depicted by the left column images, and the PiPS DLP system is depicted by the right column images. For the PiPS approach, the visualization is augmented with the egocylindrical image, which contains memory of historical depth information to reduce the effect of the sensor's limited field of view. The wall data points to the left of the robot are colored red, indicating close proximity to the robot. The other obstacles in front of, to the right of, and behind the robot are colored green to blue, indicating larger distances from the robot. Also depicted is the egocircle (colored blue) which shows that the navigation system has good knowledge of the local surroundings for informing the path scoring component of the local planner.

Failure Cases

Next, we explore the failure cases for the PiPS DLP navigation method. The two failure case types experienced were bumper collisions and aborts.

Bumper collisions occur much more frequently with the Pioneer than with the Turtlebot. The main source of bumper collisions with the Pioneer is rotating in place near an obstacle since unlike with the Turtlebot this can cause the Pioneer to go from a noncolliding state to a colliding state. This happens most frequently if a laser unsafe obstacle is directly on the global path. Such obstacles are represented in the egocylindrical model but not in the egocircle. As a result, evaluation of the cost functions prioritizes trajectories that stick close to the global path (see Fig.22a). The trajectory accepted by collision checking will therefore be the one that gets the robot as close to the obstacle as possible the obstacle. Similar behavior is exhibited by the Turtlebot, but the Turtlebot is able to safely rotate in place to follow a replanned global path while the Pioneer cannot (see Fig.22b). Even if the local planner correctly concludes that it cannot safely turn, this may result in the execution of a recovery

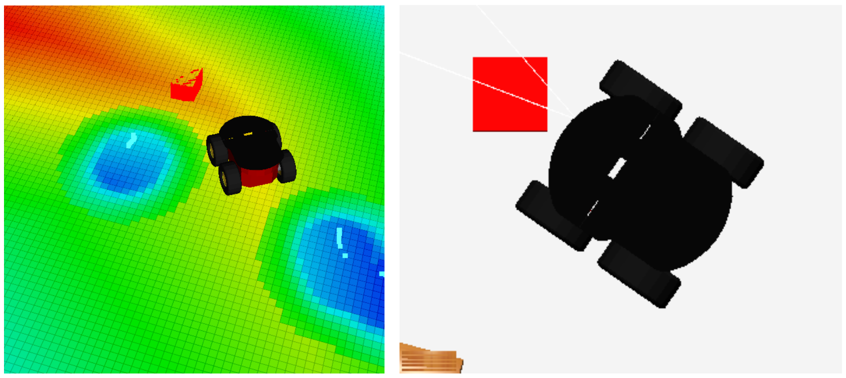

(a) Total Costs (b) Too Close

Fig. 22 (a) Visualization of approximate total costs associated with trajectories ending at each point on the grid (red=low, blue=high). A laser unsafe obstacle can be seen in front of the robot (colored red). (b) Visualization of robot's state a short time later; attempts to turn will result in collision.

behavior and still result in a collision. Incorporating collision checking information into the egocircle should help to prevent the planner from taking the robot so close to obstacles. Another option may be to permit the robot to drive backwards if it is unable to go forwards or turn in place.

Local planning can also fail by getting stuck in a local minimum. Figure 23 depicts a scenario where the Turtlebot must pass between two obstacles in order to follow the global path (brown). As shown by the color coded total cost values (red=low, blue=high), there is a local minimum on the near side of the obstacles that prevents the robot from traveling into the gap. Since the gap is sufficiently wide for the Turtlebot to safely enter it, global replanning does not provide an alternative path and navigation ultimately aborts. Tuning cost function parameters is only a partial solution since different situations may require different sets of parameters to achieve the desired behavior [61]. Incorporating concepts from gap-based approaches may help to prevent problems related to local minima.

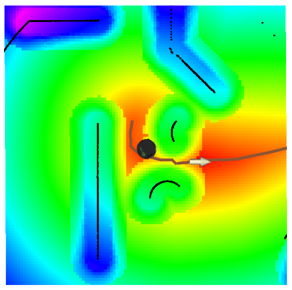


Fig. 23 Approximate total cost values of a scenario with a local minimum (red=low, blue=high). Also visualized: local goal (tan arrow), global path (brown curve), egocircle (black points).

7.4 Implementation using Stereo Camera

We now explore the performance of PiPS DLP when using a stereo camera. As mentioned in §5.4, the depth images required by the PiPS system can be produced by stereo matching algorithms. For these experiments, a stereo camera is attached 20cm

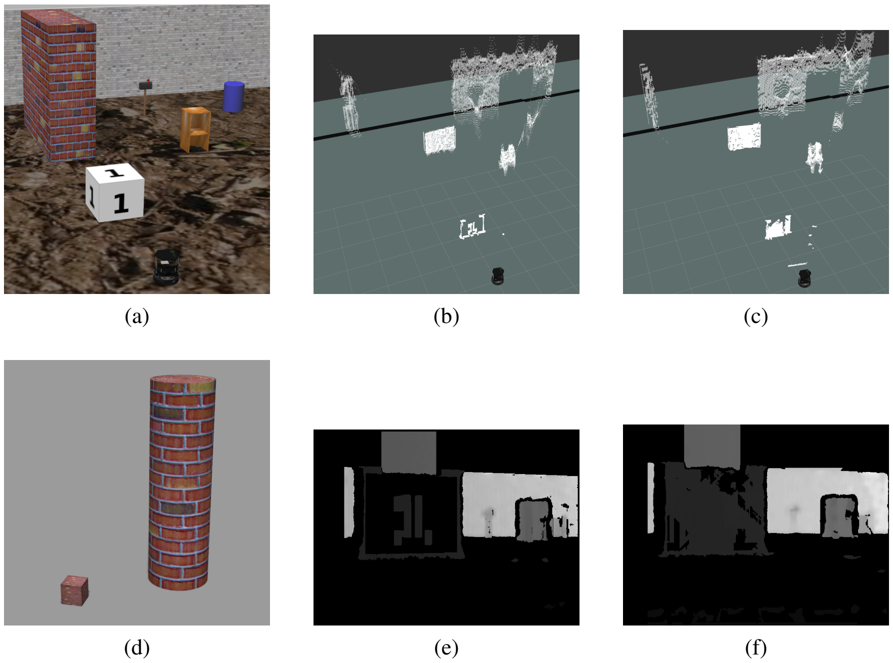


Fig. 24 Visualization of stereo implementation. (a) shows a simulated world with textured ground plane and Turtlebot; (d) shows the textured box and cylinder obstacles used in experiments; (b) and (e) depict the pointclouds and depth image generated with the traditional block matching (BM) method; (c) and (f) depict the pointclouds and depth image generated with the semi-global block matching (SGBM) approach.

in front of the Kinect on the Turtlebot and at the same position as the Realsense on the Pioneer. A stereo camera is simulated using *gazebo_ros_multicamera* Gazebo plugin with 7cm baseline, 60° field of view, 640x480 resolution and 30Hz frame rate.

The ROS stereo image processing package [115] generates disparity maps from stereo image pairs. Depth images can be easily computed from these disparity maps per Eq.20. Traditional block matching (BM) and semi-global block matching (SGBM) are both implemented in the ROS package. Figure 24 shows the results of these methods on a simulated scene after parameter tuning. The first column (Fig.24a & 24d) depicts the simulated scene. In order to improve the performance of stereo matching, texture has been added to the ground plane as well as to the randomly placed obstacles used in the experiments. The middle column (Fig.24b & 24e) shows the stereo matching results from BM for the scene in Fig.24a. The right column (Fig.24c & 24f) displays the results from SGBM for the same scene. BM and SGBM represent different trade-offs between speed and accuracy in stereo processing. BM is significantly faster than SGBM (approximately 25ms vs 120ms on the test machine), potentially enabling faster planning rates. However, the additional

Table 9 Results for sector world with laser safe obstacles using stereo implementation

Sector world with laser safe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	100%	41.90s	19.63m	0% / 0% / 0%
PiPS DLP	90%	44.85s	19.53m	0% / 10% / 0%
Pioneer				
Baseline DLP	100%	44.04s	20.00m	0% / 0% / 0%
PiPS DLP	82%	46.49s	19.74m	4% / 14% / 0%

Table 10 Results for sector world with laser safe and randomly placed laser unsafe obstacles using stereo implementation

Sector world with laser unsafe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	34%	41.90s	19.63m	66% / 0% / 0%
PiPS DLP	72%	51.58s	19.83m	6% / 22% / 0%
Pioneer				
Baseline DLP	22%	443.45s	20.44m	76% / 2% / 0%
PiPS DLP	74%	56.69s	20.61m	8% / 18% / 0%

processing performed by SGBM results in smoother, higher quality depth estimates. The difference is especially apparent when looking at each algorithm's depth estimate of the white cube in Fig.24a: BM was only able to estimate depth along the edges of the cube and the number '1' printed on it while SGBM was able to generate an accurate estimate for most of the face of the cube. Since the local planning rate used in the previous experiments was only 5Hz, the superior quality of SGBM's depth images outweighs its longer processing time.

Apart from the depth image (and consequently the virtual laser scan) being derived from stereo matching rather than a depth camera, the stereo implementation experiments are identical to the previous experiments. The experiment results are shown in Tables 9-12. In all cases, the success rate of the stereo implementation is equal or slightly lower than the success rate of the corresponding depth image implementation. The stereo implementation is vulnerable to all of the failure cases described in §7.3 in addition to stereo-related sources of failure (noise, occlusion, featureless surfaces, etc.).

8 Conclusion

Modern hierarchical navigation methods mostly rely on laser scan sensor measurements due to the computational cost of processing the depth or range imagery signals generated from contemporary sensors. Approaches geared towards resolving this problem rely on data structures that are efficient for low resolution imagery but do

Table 11 Results of campus world with randomly placed laser safe and unsafe obstacles using stereo implementation

Campus world with randomly selected laser safe and unsafe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	64%	38.58s	18.43m	34% / 2% / 0%
PiPS DLP	84%	49.83s	20.67m	8% / 8% / 0%
Pioneer				
Baseline DLP	46%	41.09s	19.50m	50% / 4% / 0%
PiPS DLP	86%	48.10s	20.88m	6% / 8% / 0%

Table 12 Results of office world with randomly placed laser safe and unsafe obstacles using stereo implementation

Office world with randomly selected laser safe and unsafe obstacles				
Approach	Success Rate	Completion Time	Path Length	Failures (BC / AB / TO)
Turtlebot				
Baseline DLP	72%	101.09s	60.50m	28% / 0% / 0%
PiPS DLP	88%	115.52s	48.38m	4% / 8% / 0%
Pioneer				
Baseline DLP	66%	100.55s	75.94m	34% / 0% / 0%
PiPS DLP	90%	104.33s	70.63m	4% / 4% / 2%

not scale well for higher resolution imagery. Modifying the internal world representation of the local planner to a viewer-centric or perception-space world representation avoids the cost of mapping the data to data structures with poor scaling and grants linear scaling properties as a function of the image resolution. A local planning pipeline for trajectory scoring and collision checking using perception space has the potential to replace the existing laser scan inspired strategies while preserving real-time operation. This chapter described a set of modifications employing perception space for a classical hierarchical navigation system. When evaluated on a described navigation benchmark, the perception space navigation system had comparable or better performance to the original laser scan implementation. Deficiencies of the system were found to be a result of the local planner scoring system rather than of the perception space modifications. Analysis and improvement of the scoring functions and the recovery behaviors should resolve the identified issues. Alternatively, exploring perception-space implementations of other local planning strategies might lead to improved performance. Future work aims to do so.

References

1. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, algorithms, and Implementation*. MIT Press, 2005.
2. S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
3. M. Ivanov, L. Lindner, O. Sergiyenko, J. C. Rodríguez-Quiñonez, W. Flores-Fuentes, and M. Rivas-Lopez, *Mobile Robot Path Planning Using Continuous Laser Scanning*. IGI

Global, 2019, pp. 338–372.

4. P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.
5. E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: http://dx.doi.org/10.1007/BF01386390
6. A. T. Stentz, “Optimal and Efficient Path Planning for Partially-Known Environments,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA ’94)*, vol. 4, May 1994, pp. 3310–3317.
7. S. Koenig and M. Likhachev, “Fast replanning for navigation in unknown terrain,” *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, June 2005.
8. M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, “Anytime search in dynamic graphs,” *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437020800060X
9. F. M. García, M. Kapadia, and N. I. Badler, “Gpu-based dynamic search on adaptive resolution grids,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 1631–1638.
10. E. G. Tsardoulias, A. Iliakopoulou, A. Kargakos, and L. Petrou, “A Review of Global Path Planning Methods for Occupancy Grid Maps Regardless of Obstacle Density,” *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 829–858, Dec. 2016.
11. O. Takahashi and R. J. Schilling, “Motion planning in a plane using generalized Voronoi diagrams,” *IEEE Transactions on Robotics and Automation*, vol. 5, no. 2, pp. 143–150, April 1989.
12. L. E. Kavraki, M. N. Kolountzakis, and J. . Latombe, “Analysis of probabilistic roadmaps for path planning,” *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, Feb 1998.
13. R. Bohlin and L. E. Kavraki, “Path planning using lazy PRM,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, April 2000, pp. 521–528 vol.1.
14. K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou, “Anytime dynamic path-planning with flexible probabilistic roadmaps,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 2372–2377.
15. J. van den Berg, D. Ferguson, and J. Kuffner, “Anytime path planning and replanning in dynamic environments,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 2366–2371.
16. E. Frazzoli, M. A. Dahleh, and E. Feron, “Real-Time Motion Planning for Agile Autonomous Vehicles,” *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2015/05/29 2002. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/2.4856
17. J. J. Kuffner and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, April 2000, pp. 995–1001 vol.2.
18. S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: http://ijr.sagepub.com/content/30/7/846.abstract
19. Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, “Motion planning for urban driving using rrt,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2008, pp. 1681–1686.
20. D. Ferguson, N. Kalra, and A. Stentz, “Replanning with RRTs,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 1243–1248.
21. M. Zucker, J. Kuffner, and M. Branicky, “Multipartite RRTs for Rapid Replanning in Dynamic Environments,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 1603–1609.

22. J. Bruce and M. Veloso, “Real-time randomized path planning for robot navigation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, Sept 2002, pp. 2383–2388 vol.3.
23. M. Otte and E. Frazzoli, *RRT X : Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles*. Cham: Springer International Publishing, 2015, pp. 461–478. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16595-0_27
24. M. Pivtoraiko and A. Kelly, “Kinodynamic motion planning with state lattice motion primitives,” in *Proceedings of the IEEE International Conference on Intelligent Robotic and Systems*, 2011.
25. K. Hauser, T. Bretl, K. Harada, and J.-C. Latombe, “Using motion primitives in probabilistic sample-based planning for humanoid robots,” in *Algorithmic Foundation of Robotics VII*, ser. Springer Tracts in Advanced Robotics, S. Akella, N. Amato, W. Huang, and B. Mishra, Eds. Springer Berlin Heidelberg, 2008, vol. 47, pp. 507–522.
26. E. Frazzoli, M. Dahleh, and E. Feron, “Maneuver-based motion planning for nonlinear systems with symmetries,” *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.
27. I. A.Ş., M. Moll, and L. Kavraki, “The open motion planning library,” *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012, http://ompl.kavrakilab.org.
28. A. A. Paranjape, K. C. Meier, X. Shi, S. Chung, and S. Hutchinson, “Motion primitives and 3-d path planning for fast flight through a forest,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 2940–2947.
29. O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, March 1985, pp. 500–505.
30. E. Rimon and D. E. Koditschek, “Exact robot navigation using artificial potential functions,” *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.
31. O. Arslan and D. Koditschek, “Exact robot navigation using power diagrams,” in *IEEE International Conference on Robotics and Automation*, 2016, pp. 1–8.
32. N. P. Hyun, E. I. Verriest, and P. A. Vela, “Optimal obstacle avoidance trajectory generation using the root locus principle,” in *IEEE Conference on Decision and Control*, 2015, pp. 626–631.
33. J. Sethian, *Level Sets Methods and Fast Marching Methods*. Cambridge University Press, 1999.
34. S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2003.
35. M. Kelly, “An introduction to trajectory optimization: How to do your own direct collocation,” *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.
36. I. M. Ross and M. Karpenko, “A review of pseudospectral optimal control: From theory to flight,” *Annual Reviews in Control*, vol. 36, no. 2, pp. 182 – 197, 2012.
37. J. Andersson, J. Gillis, G. Horn, J. Rawlings, and M. Diehl, “CasADi: a software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, Jul 2018.
38. R. Deits and R. Tedrake, “Efficient mixed-integer planning for UAVs in cluttered environments,” in *IEEE International Conference on Robotics and Automation*, 2015, pp. 42–49.
39. N. Hyun, P. Vela, and E. Verriest, “A new framework for optimal path planning of rectangular robots using a weighted l_p norm,” *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1460–1465, 2017.
40. M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, “Continuous-time Gaussian process motion planning via probabilistic inference,” *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.
41. Q. Pham, “A general, fast, and robust implementation of the time-optimal path parameterization algorithm,” *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, 2014.
42. J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization.” in *Robotics: science and systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.
43. J. Borenstein and Y. Koren, “Real-time obstacle avoidance for fast mobile robots,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, Sep 1989.

44. ——, “The vector field histogram-fast obstacle avoidance for mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, June 1991.
45. I. Ulrich and J. Borenstein, “Vfh/sup */: local obstacle avoidance with look-ahead verification,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3, April 2000, pp. 2505–2511 vol.3.
46. K. Nepal, A. Fine, N. Imam, D. Pietrocola, N. Robertson, and D. J. Ahlgren, “Combining a modified vector field histogram algorithm and real-time image processing for unknown environment navigation,” in *Intelligent Robots and Computer Vision XXVI: Algorithms and Techniques*, vol. 7252. International Society for Optics and Photonics, 2009, p. 72520G.
47. and, “Vph: a new laser radar based obstacle avoidance method for intelligent mobile robots,” in *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No.04EX788)*, vol. 5, June 2004, pp. 4681–4685 Vol.5.
48. J. Gong, Y. Duan, Y. Man, and G. Xiong, “Vph+: An enhanced vector polar histogram method for mobile robot obstacle avoidance,” in *2007 International Conference on Mechatronics and Automation*, Aug 2007, pp. 2784–2788.
49. J. Minguez and L. Montano, “Nearness diagram navigation (nd): a new real time collision avoidance approach,” in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, vol. 3, Oct 2000, pp. 2094–2100 vol.3.
50. ——, “Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios,” *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, Feb 2004.
51. J. W. Durham and F. Bullo, “Smooth nearness-diagram navigation,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2008, pp. 690–695.
52. J. Minguez, L. Montano, T. Simeon, and R. Alami, “Global nearness diagram navigation (gnd),” in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 1, May 2001, pp. 33–39 vol.1.
53. M. Mujahad, D. Fischer, B. Mertsching, and H. Jaddu, “Closest Gap based (CG) reactive obstacle avoidance Navigation for highly cluttered environments,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2010, pp. 1805–1812.
54. M. Mujahed, D. Fischer, and B. Mertsching, “Safe Gap based (SG) reactive navigation for mobile robots,” in *2013 European Conference on Mobile Robots (ECMR)*, Sept 2013, pp. 325–330.
55. V. Sezer and M. Gokasan, “A novel obstacle avoidance algorithm: "Follow the Gap Method",” *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889012000838
56. M. Mujahed and B. Mertsching, “The admissible gap (AG) method for reactive collision avoidance,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1916–1921.
57. R. Bauer, W. Feiten, and G. Lawitzky, “Steer angle fields: An approach to robust manoeuvring in cluttered, unknown environments,” *Robotics and Autonomous Systems*, vol. 12, no. 3, pp. 209 – 212, 1994.
58. W. Feiten, R. Bauer, and G. Lawitzky, “Robust obstacle avoidance in unknown and cramped environments,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, May 1994, pp. 2412–2417 vol.3.
59. J. Buhmann, W. Burgard, A. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun, “The Mobile Robot RHINO,” *AI Magazine*, vol. 16, no. 1, 1995.
60. D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *Robotics Automation Magazine, IEEE*, vol. 4, no. 1, pp. 23–33, Mar 1997.
61. C. Stachniss and W. Burgard, “An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2002, pp. 508–513 vol.1.
62. O. Brock and O. Khatib, “High-speed navigation using the global dynamic window approach,” in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 341–346 vol.1.

63. E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The Office Marathon: Robust navigation in an indoor office environment," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 300–307.
64. R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, Apr 1996, pp. 3375–3382 vol.4.
65. N. Y. Ko and R. Simmons, "The lane-curvature method for local obstacle avoidance," in *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, vol. 3, Oct 1998, pp. 1615–1621 vol.3.
66. S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, May 1993, pp. 802–807 vol.2.
67. M. Khatib, "Sensor-based motion control for mobile robots," *LAAS-CNRS: Toulouse, France*, 1996.
68. C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK 2012; 7th German Conference on Robotics*, May 2012, pp. 1–6.
69. C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889016300495
70. L. Lindner, O. Sergiyenko, M. Rivas-López, D. Hernández-Balbuena, W. Flores-Fuentes, J. R.-Q. nonez, F. Murrieta-Rico, M. Ivanov, V. Tyrsa, and L. Básaca-Preciado, "Exact laser beam positioning for measurement of vegetation vitality," *Industrial Robot: The International Journal of Robotics Research and Application*, vol. 44, no. 4, pp. 532–541, 2017.
71. O. Sergiyenko, M. Ivanov, V. Tyrsa, V. Kartashov, M. Rivas-López, D. Hernández-Balbuena, W. Flores-Fuentes, J. R.-Q. nonez, J. Nieto-Hipólito, W. Hernandez, and A. Tchernykh, "Data transferring model determination in robotic group," *Robotics and Autonomous Systems*, vol. 83, pp. 251 – 260, 2016.
72. M. Ivanov, O. Sergiyenko, V. Tyrsa, P. Mercorelli, V. Kartashov, W. Perdomo, S. Sheiko, and M. Kolendovska, "Individual scans fusion in virtual knowledge base for navigation of mobile robotic group with 3d tvs," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, 10 2018, pp. 3187–3192.
73. D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, Nov 2012, pp. 692–697.
74. D. Murray and C. Jennings, "Stereo vision based mapping and navigation for mobile robots," *IEEE Int Conf Robotics Automation*, vol. 2, pp. 1694 – 1699 vol.2, 05 1997.
75. J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 1697–1702.
76. A. J. Barry, P. R. Florence, and R. Tedrake, "High-speed autonomous obstacle avoidance with pushbroom stereo," *Journal of Field Robotics*, vol. 35, no. 1, pp. 52–68, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21741
77. K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," in *In Proc. of the ICRA 2010 workshop*, 2010.
78. K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller, and M. Suppa, "Stereo vision based indoor/outdoor navigation for flying robots," *IEEE International Conference on Intelligent Robots and Systems*, pp. 3955–3962, 2013.
79. B. T. Lopez and J. P. How, "Aggressive 3-D collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5759–5765.

80. L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3242–3249.
81. T. Cao, Z. Xiang, and J. Liu, "Perception in disparity: An efficient navigation framework for autonomous vehicles with stereo cameras," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2935–2948, Oct 2015.
82. L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3242–3249.
83. K. McGuire, G. de Croon, C. De Wagter, K. Tuyls, and H. Kappen, "Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1070–1076, April 2017.
84. S. Hrabar, G. S. Sukhatme, P. Corke, K. Usher, and J. Roberts, "Combined optic-flow and stereo-based navigation of urban canyons for a uav," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug 2005, pp. 3309–3316.
85. T. Cao, Z. Xiang, and J. Liu, "Perception in disparity: An efficient navigation framework for autonomous vehicles with stereo cameras," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2935–2948, Oct 2015.
86. M. W. Otte, S. G. Richardson, J. Mulligan, and G. Grudic, "Local path planning in image space for autonomous robot navigation in unstructured environments," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2007, pp. 2819–2826.
87. J. Cutting, P. Vishton, and P. Braren, "How we avoid collisions with stationary and moving obstacles," *Psychological Review*, vol. 102, no. 4, pp. 627–651, Oct 1995.
88. B. R. Fajen, "Guiding locomotion in complex, dynamic environments," *Frontiers in Behavioral Neuroscience*, vol. 7, JUL 19 2013.
89. G. Vallar, E. Lobel, G. Galati, A. Berthoz, L. Pizzamiglio, and D. Le Bihan, "A fronto-parietal system for computing the egocentric spatial frame of reference in humans," *Experimental Brain Research*, vol. 124, no. 3, pp. 281–286, Jan 1999. [Online]. Available: https://doi.org/10.1007/s002210050624
90. M. R. Dillon, A. S. Persichetti, E. S. Spelke, and D. D. Dilks, "Places in the brain: Bridging layout and object geometry in scene-selective cortex," *Cerebral Cortex*, vol. 28, no. 7, pp. 2365–2374, 2018.
91. D. D. Dilks, J. B. Julian, A. M. Paunov, and N. Kanwisher, "The occipital place area is causally and selectively involved in scene perception," *Journal of Neuroscience*, vol. 33, no. 4, pp. 1331–1336, 2013.
92. M. R. Greene and A. Oliva, "Recognition of natural scenes from global properties: Seeing the forest without representing the trees," *Cognitive Psychology*, vol. 58, no. 2, pp. 137 – 176, 2009.
93. M. Bonner and E. RA, "Coding of navigational affordances in the human visual system," *Proceedings of the National Academy of Sciences*, vol. 114, no. 18, pp. 4793–4798, 2017.
94. D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. MIT Press, 1982.
95. G. Galati, E. Lobel, G. Vallar, A. Berthoz, L. Pizzamiglio, and D. Le Bihan, "The neural basis of egocentric and allocentric coding of space in humans: a functional magnetic resonance study," *Experimental Brain Research*, vol. 133, no. 2, pp. 156–164, Jul 2000. [Online]. Available: https://doi.org/10.1007/s002210000375
96. R. F. Wang and E. S. Spelke, "Updating egocentric representations in human navigation," *Cognition*, vol. 77, no. 3, pp. 215–250, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010027700001050
97. H. J. Spiers and E. A. Maguire, "A navigational guidance system in the human brain," *Hippocampus*, vol. 17, no. 8, pp. 618–626. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hipo.20298
98. R. A. Epstein, "Parahippocampal and retrosplenial contributions to human spatial navigation," *Trends in Cognitive Sciences*, vol. 12, no. 10, pp. 388–396, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S136466130800199X

99. A. A. Wilber, B. J. Clark, T. C. Forster, M. Tatsuno, and B. L. McNaughton, "Interaction of Egocentric and World-Centered Reference Frames in the Rat Posterior Parietal Cortex," *Journal of Neuroscience*, vol. 34, no. 16, pp. 5431–5446, 2014. [Online]. Available: http://www.jneurosci.org/content/34/16/5431
100. J. Smith and P. Vela, "Planning in perception space," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 6204–6209.
101. P. Felzenszwalb and D. Huttenlocher, "Distance transforms of sampled functions," *Theory of Computing*, vol. 8, no. 19, pp. 415–428, 2012.
102. R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.
103. OctoMap, "Github - octomap/octomap_mapping," Oct 2017. [Online]. Available: https://github.com/OctoMap/octomap_mapping
104. J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 3859–3866.
105. A. F. L. M. R. Brockers, "Stereo vision-based obstacle avoidance for micro air vehicles using an egocylindrical image space representation," vol. 9836, 2016, pp. 9836–9836–7. [Online]. Available: http://dx.doi.org/10.1117/12.2224695
106. M. Asada, T. Tanaka, and K. Hosoda, "Adaptive binocular visual servoing for independently moving target tracking," *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3, no. April 2016, pp. 2076–2081, 1999. [Online]. Available: http://ieeexplore.ieee.org/document/846335/
107. D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *International Journal of Computer Vision*, vol. 47, no. 1, pp. 7–42, 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1014573219977
108. H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, June 2005, pp. 807–814 vol. 2.
109. S. Jin, J. Cho, X. D. Pham, K. M. Lee, S. Park, M. Kim, and J. W. Jeon, "Fpga design and implementation of a real-time stereo vision system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, Jan 2010.
110. Y. Li, C. Yang, W. Zhong, Z. Li, and S. Chen, "High throughput hardware architecture for accurate semi-global matching," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 641–646.
111. D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez, J. C. Moure, and A. M. López, "Embedded real-time stereo estimation via semi-global matching on the gpu," *Procedia Computer Science*, vol. 80, pp. 143–153, 2016.
112. C. Sprunk, J. Röwekämper, G. Parent, L. Spinello, G. D. Tipaldi, W. Burgard, and M. Jalobeanu, "An experimental protocol for benchmarking robotic indoor navigation," in *Experimental Robotics*. Springer, 2016, pp. 487–504.
113. J. Smith, J. Hwang, and P. Vela, "Benchmark worlds for testing autonomous navigation algorithms," 2018, [Repository]. [Online]. Available: http://github.com/ivalab/NavBench
114. C. Rockey, "depthimage_to_laserscan," 2014, [Repository]. [Online]. Available: https://github.com/ros-perception/depthimage_to_laserscan
115. P. Mihelich, K. Konolige, and J. Leibs, "Github - ros-perception/image_pipeline/stereo_image_proc." [Online]. Available: https://github.com/ros-perception/image_pipeline.git