VISION-BASED AUTONOMOUS NAVIGATION IN MEDIUM LEVEL REPRESENTATION

A Thesis
Presented to
The Academic Faculty

By

Jin Ha Hwang

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2017

VISION-BASED AUTONOMOUS NAVIGATION IN MEDIUM LEVEL REPRESENTATION

Approved by:

Dr. Patricio A. Vela, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Fumin Zhang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Anthony J. Yezzi
School of Electrical and Computer Engineering
Georgia Institute of Technology

This thesis is dedicated to Jooyoung Kim, who has been my life since I met her.

ACKNOWLEDGEMENTS

First and foremost, I would first like to thank my thesis advisor Dr. Patricio A. Vela of the School of Electrical and Computer Engineering at Georgia Institute of Technology. He consistently assisted and steered me in the right direction whenever he thought I needed it. I sincerely hope I continue to have opportunities to interact with him for the rest of my career.

My sincere thanks also go to Justin Smith at IVALab who supported me throughout various research projects and my thesis with insightful comments and discussions. I cannot adequately express how thankful I am. This thesis would not have been possible without his intellectual contribution and guidance throughout my entire research period.

I also thank the rest of my thesis committee members, Dr. Fumin Zhang and Dr. Anthony Yezzi for sharing their precious time.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Autonomous navigation for a mobile robot is required to operate in cluttered, unstructured environment at high speeds with efficient data gathering. Given the payload constraints and long-range sensing requirements, vision-based scene analysis is the preferred sensing modality for a modern navigation system. For outdoor navigation, stereo vision is favored by reasons of improving detection range by increasing the views of the scene and the indirect access to depth information.

However, state of the art approach uses stereo camera observations by converting disparity images to a world representation such as 3D point cloud and 2D occupancy grid and fail to deal with sensor noises [1]. The computational burden of performing dense stereo matching and updating the observation in the world representation and the difficulty in dealing with sensor error force modern approaches to update the world representation on a per frame basis, which often becomes a factor of reducing a degree of autonomy for a navigation task [2]. Moreoever, the observation update in the 2D world representation often requires a simplification of a geometry of an object as a circle [3] or a rectangle [4, 5] for the obstacle expansion. Overly inflated region due to the simplified geometry causes the navigation system to negotiate by performing overly conservative path planning.

In this paper, we propose an alternative scene perception and planning approach in a medium level representation called Stixel World for stereo cameras. Instead of converting the local scene observation into the world representation, obstacle detection and path planning computation remains in the perception space. We use a method to construct the medium level representation by detecting every possible vertical obstacle for each column in the image. We construct the Stixel representation using a reduced detection window for the Stixels by ground plane estimation. Instead of computing the full disparity map, we utilize a cost volume matrix approach. Also, we propose a method that directly projects a robot model into the perception space so that the 3D physical geometry of the robot model

does not need to be simplified unlike modern perception-based local planning approaches that perform obstacle expansion in the image space [6, 7]. Furthermore, we propose a method to integrate this local/reactive obstacle avoidance controller with a global planner for navigating to a provided destination both safely and efficiently.

We demonstrate these capabilities on a simulated mobile robot in many environment scenarios for quantitative evaluation. Then, we also qualitatively explore limitations and possibilities of the vision-based autonomous navigation in the medium level representation by implementing our approach on a real mobile robot. From experimental results, we show the robustness and effectiveness of our method by comparing to other traditional 2D Cartesian-based navigation planners as well as the depth image-based path planner.

CHAPTER 1
INTRODUCTION

Recently, many vision-based autonomous navigation methods have been proposed as an intelligent mobile robot application. While the autonomous navigation requires a relation between the perception of the environment and a low-level robot operation, the visual sensors are especially efficient for this task by providing comprehensive understandings of the local scene than range scanning sensors such as LRF and sonar sensors.

An issue that arises with vision-based obstacle avoidance is that noise in the vision source can cause an inaccurate detection of obstacles that leads to potential collisions. The navigation planner has to compromise this issue by selecting a path candidate that does not have obstacle present as well as less noise along the path, which causes overall navigation overly conservative with longer travel time [8, 9]. Hence, robust and safe obstacle avoidance has remained an active research area for several decades [10].

Achieving safe, autonomous, fast control of traditional navigation planners in unstructured environments presents two challenges. One is the need for a rapid sensing of the local scene to allow for adequate time to detect and avoid obstacles. Another is the need for a fast and accurate update of the world representation based on the newly collected information. Even though an infrared-based depth camera provides relatively accurate depth information of the scene, it does not work adequately in the outdoor scene because the IR structured lighting pattern gets lost in ambient IR. An alternative is a Time-of-Flight (ToF) camera that gives the depth information based on measuring the time-of-flight of a light between the camera and the subject for each point of the image. Even though the ToF camera performs well in low-textured regions, the performance of depth estimation over textured regions is still insufficient in the outdoor scene [11]. Instead, disparity-based depth estimation using a stereo camera is commonly performed for the autonomous navigation task. The stereo

camera offers a low weight, long range sensing that can be easily mounted on any mobile platform at the cost of increased computation.

However, fulfilling the two challenges mentioned above requires a significant computational burden on the navigation system because the traditional vision-based navigation planners generally perform a scene analysis and path planning separately in different representation spaces and the latter cannot be done without updated information in the world representation. Also, using estimated raw disparity data as the primary source for the navigation causes a lot of noise due to sensor error, which forces the navigation system to update the world representation as often as possible to correct noises [2]. Long computational time in the observation update process causes high latencies in an end-to-end global navigation pipeline, which can decrease the planning frequency and possibly overall performance in navigation tasks.

In this thesis, we propose an alternative approach to integrate the two stages to be performed in the perception space so that the navigation system can perform more robust scene analysis and obstacle avoidance. From raw stereo image pairs, we construct the medium-level representation called Stixel World. We increase the speed of the Stixel representation construction by reducing a detection window of the Stixels based on ground plane estimation. Unlike other contemporary perception-based path planning approaches that expand obstacles in the perception space, we directly project the robot model into the perception space so that the planner does not need to simplify the 3D physical geometry of the mobile robot. Then, we show that our approach can be integrated with a traditional global planner framework by proposing a new end-to-end global navigation framework.

This thesis is ordered as follows. Chapter 2 presents a short background of the related work. Chapter 3 describes the construction of the Stixel representation from a rectified stereo pair by extending [12]. Chapter 4 describes the scene analysis for the collision checking and local path planning in the perception space. Chapter 5 describes the integration of the local reactive obstacle avoidance approach with the global navigation frame-

work. Chapter 6 presents various experimental results and limitations of our approach. Finally, this paper concludes with remaining works.

CHAPTER 2

BACKGROUND

To estimate disparity from a raw stereo image pair, stereo matching should be performed to find the corresponding points in the image pair. There have been an extensive search of stereo matching algorithms such as Fast block matching [13], Semi-Global Block Mathing (SGBM) [14], and Symmetric stereo matching [15]. From estimated disparities, traditional stereo vision-based path planning algorithms commonly represent data in the world representation and plan the path.

A 2D occupancy grid is one of popular representation that the traditional navigation system performs path planning [16, 1, 17, 2]. For 3D navigation on flying vehicles such as MAVs, a 3D occupancy grid such as OctoMaps [18] is frequently used due to its efficient structure for the 3D occupancy mapping. A voxel grid is another popular modality in the world space that represents an object as a collectioin of aligned boxes [5]. [19] proposes a spherical coordinate based grid mapping for autonomous navigation using stereo sensors, but requires the disparity map to be converted to 3D point cloud before mapping into the grid.

Since many navigation systems that describes the local scene in the world space use a grid-like representation, there have been a lot of attempts to employ graph-based path searching algorithms for autonomous navigation. Dynamic Window Approach [20], Elastic Band [21], Timed-Elastic Band [22], and Genetic Algorithm [23] which generally plans paths on the 2D or 3D occupancy grid have been proposed. These approaches work sufficiently, but vision-based navigation planners that employ same approach require the additional process of converting visual observations into a data type that is compatible with the grid representation such as the point cloud. This conversion process causes a long delay during the motion planning, which potentially decreases the performance of the navigation

system. Moreover, these approaches require a large local memory to update and store the information. As travel duration and distance of a navigation task increase, maintaining the scene observations in the world representation can be problematic for small mobile robots that have hardware constraints.

Instead of path planning approaches that operate in the world representation, several works propose path planning approaches in the perception space directly without the conversion process. For outdoor navigation, [6] proposes a path planning approach that performs the A* path searching algorithm in the disparity space based on the cost assigned to each pixel. However, this approach does not consider the hardware limits and configurations of the mobile vehicle and treat the image path planning same as a grid path searching, which simply connects pixel by pixel. This unclear understanding of the relationship between the path planning in the perception space and the low-level robot operation causes the vehicle to be unable to achieve a planned path in a given period. In contrast, there are alternative approaches to construct a motion library, which is a set of trajectories that the robot can achieve based on its hardware limitations. The motion library is usually constructed prior to a navigation task and then used by projecting the corresponding image coordinates in the disparity space [24, 25].

For a robot that has a non-negligible size, many works propose an expansion-based path planning approaches, which expands an obstacle boundary by the vehicle radius incorporating with the depth information, such as Configuration-Space (C-Space) expansion [26, 27]. However, most expansion-based approaches simplify the geometry of the robot, which usually negotiates the simplification by the over-expansion of the obstacle.

As opposed to the obstacle expansion approaches, [28] proposes a robot model projection-based local path planning approach in the depth space, which does not simplify the 3D geometry of the robot. However, this approach is implemented as a local reactive obstacle avoidance controller that navigates without a goal.

Besides planning in the raw image representation and the 3D world, there has been a

lack of attempts to perform the local path planning in a middle-ground between the image level and world level. However, several methods of constructing the medium-level representations are frequently proposed such as superpixel [29, 30] and 3D primitives [31]. One other alternative is called Stixel World [32].

First proposed by Bandino *et al.* as Stixel World [32], the Stixel representation is referred as the medium level representation of 3D traffic scene with a goal to bridge the gap between pixels and a 3D physical object. Stixels are represented by a set of rectangular sticks standing vertically on the ground to approximate free space in front of the vehicle as shown in Figure 2.1.

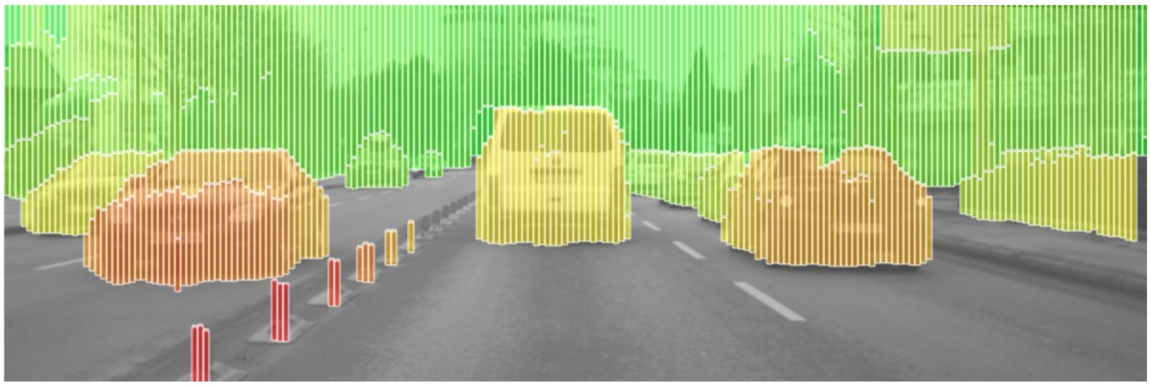


Figure 2.1: Stixel World representation

[33] proposes a multi-layer approach that allows multiple Stixels in an image column. [12] proposes Stixel computation without estimating the depth map based on an assumption that all stereo matching methods that yield a dense depth map will either use smoothing constraints or prior knowledge. [34] introduces a bottom-up stixel segmentation with object-level knowledge in a sound probabilistic fashion.

Modern Sticel construction methods implement machine learning-driven approaches such as Semantic Stixels which infers both geometric and semantic layout of a scene [35]. [36] proposes a single color camera-based Stixel representation using a deep CNN with a loss function based on a semi-discrete representation of the obstacle position probability to train the network. Stixel representation has been used for several applications such as object

detection and recognition [37, 38, 39, 40], motion estimation [41], and scene understanding [42, 43].

While there are many approaches to recognize and understand the local scene in the world representation based on the Stixels, there has been no attempts to perform the path planning in the Stixel representation directly. For a small-sized mobile robot that has hardware limitations, sophisticated scene understanding algorithms such as object recognition and segmentation from the Stixels and represent them in the world can be extremely demanding and might be unnecessary for a simple navigation task.

CHAPTER 3
STIXEL REPRESENTATION

The proposed Stixel estimation algorithm extends [12] and includes five sub-stages. From an input rectified stereo image pair, the proposed method computes the cost volume matrix of each pixel in §3.1. This cost volume reduces the search window of the Stixels by estimating a ground plane in §3.2, which is then used to estimate the Stixel disparity §3.3. We then show height estimation of the Stixel using the estimated Stixel disparity in §3.4. Throughout the paper, we assume a width of the Stixel to be one pixel for simplicity of mathematical equations. Using a broader Stixel width may reduce the overall time complexity by averaging columns of data, but reduces the spatial resolution of the Stixels as a tradeoff.

3.1 Cost Volume Computation

Given a pair of rectified stereo images, the Stixel estimator first computes a matching cost volume, $c_m(u, v, d)$, which is a cost of having a disparity d at pixel (u, v) by using a sum of absolute differences over the image channel for every pixel in the left image and every possible disparity value $d \in [0, d_{max}]$. In this paper, we denote u, v, d as a horizontal axis, vertical axis, and disparity respectively. We describe how the cost volume is computed in Algorithm 1.

The lower the cost $c_m(u, v, d)$ is, the likelihood of having the disparity value d at (u, v) is higher. This pixelwise operation can be parallelized to increase the computational speed. Most stereo matching methods include an equivalent matching step with similar or higher cost, which is usually followed by a smoothing process that dominates the computation time [44].

Algorithm 1 Cost Volume Computation

1: **procedure** COSTVOLUME(u, v)
2: $d \leftarrow 0$
3: $d_{max} \leftarrow$ Max disparity value possible
4: $I(u, v) \leftarrow$ intensity at $image(u, v)$
5: **while** $d \leq d_{max}$ **do**
6: $c_m(u, v, d) \leftarrow I_{left}(u, v) - I_{right}(u - d, v)$

3.2 Ground Plane Estimation

The Stixel estimator then finds a ground plane by exploiting a $v\text{-}disparity$ representation [45]. Since one assumption of the Stixel world is that an object stands vertically from the ground, ground estimation efficiently reduces the search space of the Stixels and speeds up the computation of following stages.

For a vehicle that equips the camera with the pitch angle of 0 or small negative (*i.e.*, camera facing straight forward or slightly downward), a horizontal vanishing line of the ground plane is guaranteed to converge at the center point of the vertical axis in the image. Therefore, the ground estimator first exclude the upper half of the image (*i.e.*, $v \in [0, \frac{width}{2}]$) from the search window.

To calculate the ground plane, we first project the cost volume matrix computed in §3.1 along the horizontal axis (u-axis) to create the $v\text{-}disparity$ representation. Each pixel in the $v\text{-}disparity$ representation contains a summed pixel cost of the uni-dimensional slice of the cost volume. The equation of $v\text{-}disparity$ computation is the following.

$$I_{v\Delta}(d, v) = \sum_{u=0}^{width} c_m(u, v, d) \tag{3.1}$$

For an image in Figure 3.1, a line that connects lowest cost at each row, $\arg\min_d I_{v\Delta}(d, v)$, has constant slope in disparity. On the other hand, there is no change in $\arg\min_d I_{v\Delta}(d, v)$ for $v \in [0, v(wall)]$ from the $v\text{-}disparity$ representation in Figure 3.2. This means that this area has the high probability of having the constant disparity value vertically, which we

assume the area as a non-ground plane. Therefore, the estimator excludes this area from the Stixel search window based on the assumption of the Stixel World where all objects stand vertically from the ground plane.

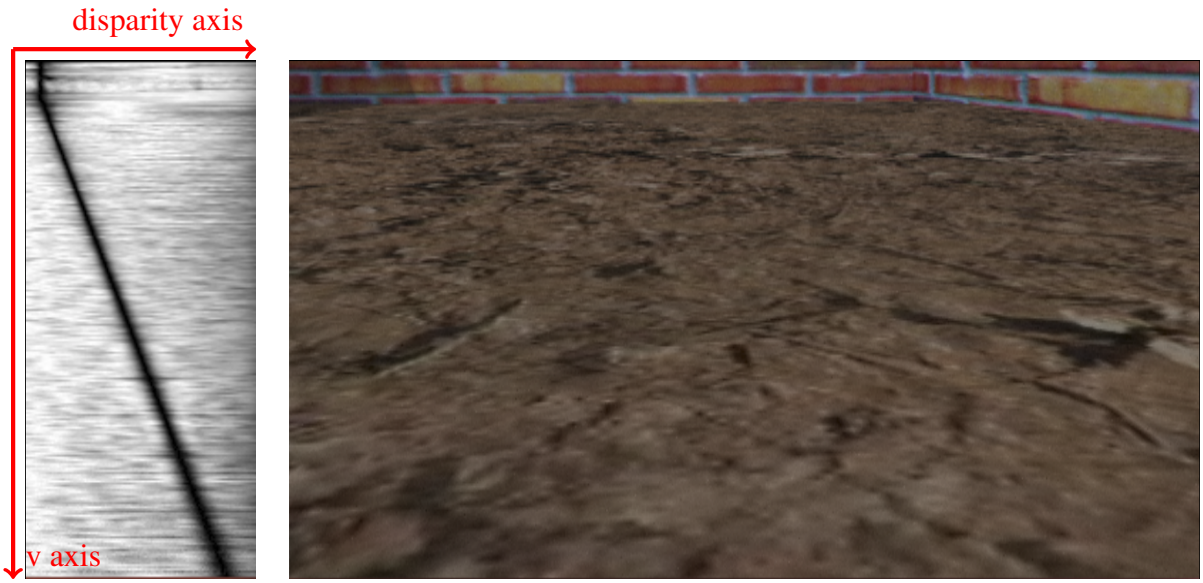


Figure 3.1: *v-disparity* representation and original image with no wall present

Figure 3.2: *v-disparity* representation and original image with a vertical wall present

3.3 Stixel Disparity Estimation

While ground plane estimation involves the projection of the cost volume along the horizontal axis, the Stixel estimator projects the cost volume matrix along the vertical axis to estimate a Stixel distance in each column of the image.

Following the approach of Kubota *et al.* [46], the estimator computes the disparity of each Stixel using 2D dynamic programming over two terms: Stixel cost $c_s(u, v)$ and smoothness term $s_s(d(u_a), d(u_b))$ where u_a and u_b are neighbouring pixels ($|u_a - u_b| = 1$). We compute an optimal disparity of the Stixel at column u, $d_s^*(u)$, by solving the 2D minimization problem using the dynamic programming in the $u\text{-}disparity$ domain.

$$d_s^*(u) = \arg\min_{d(u)} \sum_u c_s(u, d(u)) + \sum_{u_a, u_b} s_s(d(u_a), d(u_b)) \tag{3.2}$$

How each cost is defined will be explained below.

3.3.1 Stixel Cost

For each column and possible disparity $d \in [0, d_{max}]$, a Stixel cost $c_s(u, d)$ defines a likelihood of a presence of a Stixel at d in the left image. The lower the cost, the more likely that the Stixel is present at position $(u, v(d))$. The Stixel cost comprises of an object cost $c_o(u, d)$ and a ground cost $c_g(u, d)$.

$$c_s(u, d) = c_o(u, d) + c_g(u, d) \tag{3.3}$$

Object Cost

The object cost $c_o(u, d)$ describes the cost of a vertical object being present at $(u, v(d))$. For a faster and more accurate computation, we set up an upper bound of expected object height h_o. Using a predefined camera calibration matrix and the estimated ground plane from §3.2, the Stixel estimator calculates vertical image coordinates of a point above the

ground for a given height $v(h, d)$. With h_o, the estimator calculates $v(h_o, d)$ for our upper boundary of the Stixel in the image plane. If no h_o is specified, $v(h_o, d)$ should be 0 (*i.e.*, top of the image).

$$c_o(u, d) = \sum_{v=v(d)}^{v(h_o,d)} c_m(u, v, d) \tag{3.4}$$

From Figure 3.3, object costs $c_o(u, d)$ in the cost matrix at disparity value d that corresponds to the location of the obstacle are lower than other disparity values in each column. In Figure 3.3a, object costs are lower for smaller disparity values for the columns where a trash bin is located ($c_o(u(trash), d)$) while costs are evenly distributed over all disparity values for the columns where a vertical wall is located. Also, in Figure 3.3b, the disparity values that have lowest $c_o(u(coke), d)$ is generally larger than the disparity values that have lowest $c_o(u(cabinet), d)$ because the coke is located closer from the camera than the coke.

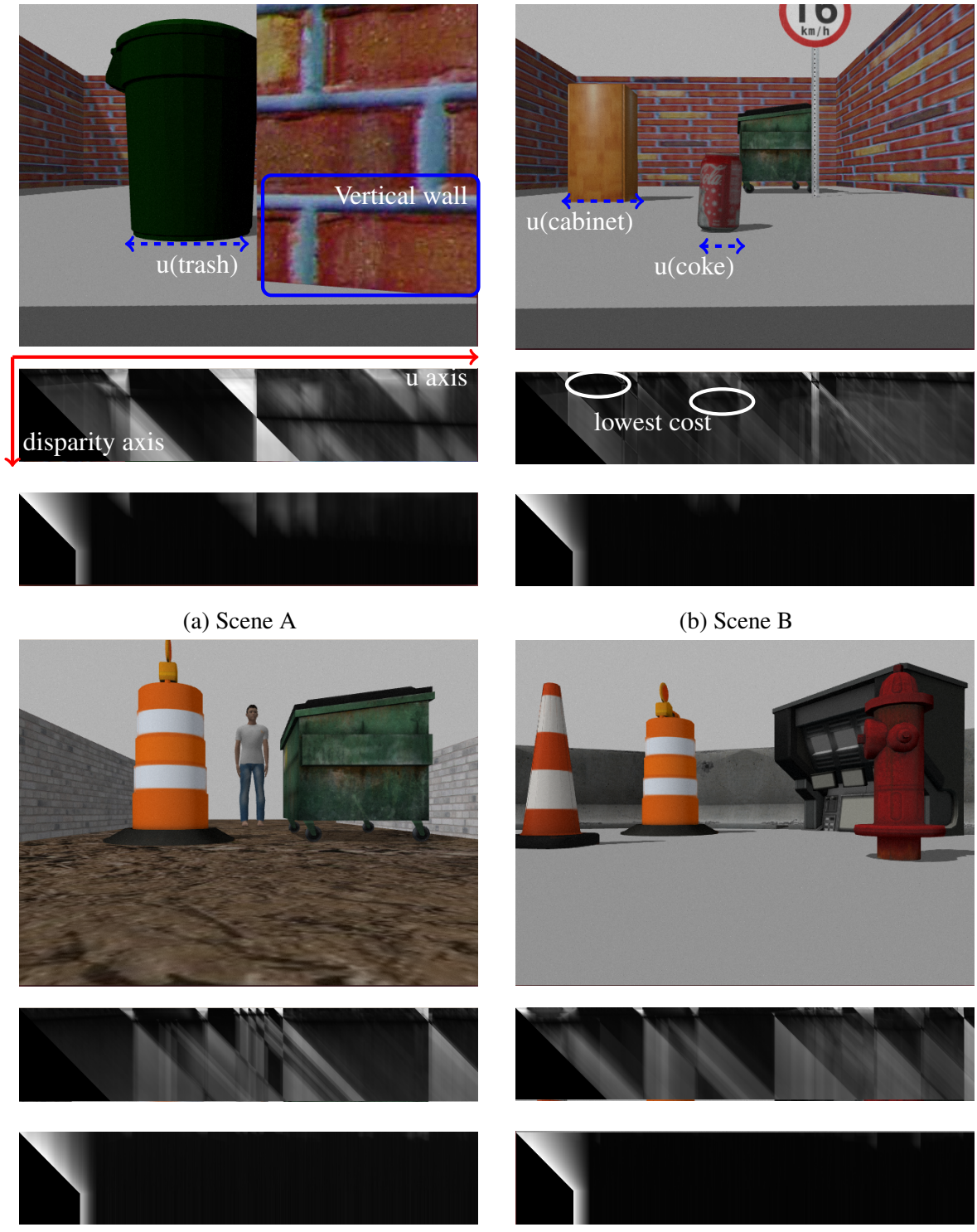
Ground Cost

The ground cost $c_g(u, d)$ describes the cost of a supporting ground being present at $(u, v(d))$. For a flat surface, it is bijective relationship between $f_{ground} : U \times V \mapsto D$. Therefore, we can map (u, d) coordinate to a point $(u, v(d))$ in the image plane using $v(d) = f^{-1}_{ground}(d)$. For the ground cost, the Stixel estimator computes a likelihood of the supporting ground plane's presence at $v = v(d)$ for each $d \in [0, d_{max}]$.

$$c_g(u, d) = \sum_{v=height_{image}}^{v(d)} c_m(u, v, f_{ground}(v)) \tag{3.5}$$

From the image in Figure 3.3a, a vertical wall is present in the ground plane search area. Therefore, ground cost matrix contains higher cost at smaller disparities for the columns where the wall is located, which indicates that this region is less likely to have the supporting ground if there is an obstacle. For both object and ground cost matrices, we discard the far left columns since they are the uncertainty due to unknown stereo matching due to the

baseline between two cameras. As the baseline if the stereo camera increases, the size of the uncertain columns will also increase.



(a) Scene A

(b) Scene B

(c) Scene C

(d) Scene D

Figure 3.3: Object cost (top) and ground cost (bottom) matrices in different local scenes

3.3.2 Smoothness Term

In a stereo vision, some portion of the objects visible in the left image can be occluded in the right image, causing a significant difference in computed disparity values. When processing the left image, the Stixel estimator excludes this area from the Stixel computation since it does not have corresponding area to match. We ensure this occlusion constraint using a smoothness term by exploiting [46].

$$s_s(d_a, d_b) = \begin{cases} \infty, \text{ if } d_a < d_b - 1 \\ c_o(u_a, u_b), \text{ if } d_a = d_b - 1 \\ 0, \text{ if } d_a > d_b - 1 \end{cases} \tag{3.6}$$

where $d_a = d(u_a)$ and $d_b = d(u_b)$ with u_a one pixel to the left of the pixel u_b. When the s_s is ∞, no Stixel distance estimate will violate the occlusion constraint.

3.3.3 Dynamic Programming

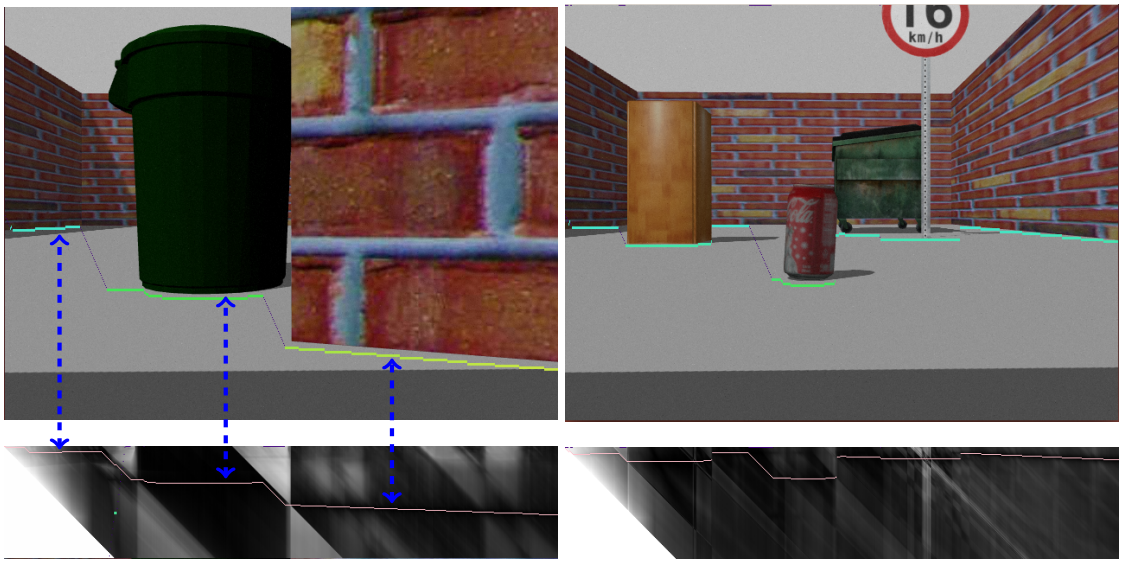
With computed Stixel cost c_s and smoothness term s_s, an optimal disparity for each Stixel, d_s^*, is computed by solving the 2D dynamic programming for the cost minimization in the $u\text{-}disparity$ domain using [46]. The Stixel estimator performs the dynamic programming calculation from the right most column to the left most column using the following recursive equations.

$$\begin{aligned} d_s^*(u_{width}) &= \underset{d(u_{width})}{\arg\min}\, c_s(u_{width}, d(u_{width})) \\ d_s^*(u-1) &= \underset{d(u-1)}{\arg\min}\, \{c_s(u-1, d(u-1)) + s_s(d(u-1), d_s^*(u))\} \end{aligned} \tag{3.7}$$

where u_{width} is the far right column of the image.

Using the $d_s^*(u)$ in the $u\text{-}disparity$ domain and predefined camera calibration matrix, the Stixel estimator computes the u-v boundary $v_{bottom}^*(u)$ that represents a bottom of each

Stixel. In Figure 3.4, we show original left images in different local scenes with the corresponding Stixel cost matrices in the u-$disparity$ domain layered with $d_s^*(u)$ along the column. We also draw the bottom of the Stixel at $(u, v(d_s^*(u)))$ in the original image for each u.



(a) Scene A

(b) Scene B

(c) Scene C

(d) Scene D

Figure 3.4: Stixel representation in the left image with corresponding Stixel cost matrix

3.4 Stixel Height Estimation

[32] uses the input depth map to compute a membership function based on the distance between the pixelwise disparities and the Stixel disparities. Since our Stixel computation does not include the full dense disparity map computation, we use an alternative approach that computes a similar membership function without estimating pixelwise disparities.

If a pixel (u, v) in the image belongs to a given disparity d, we expect the cost volume function of neighbouring pixels with d, $c_m(u, v, d_{surr})$ to be a local minima. If a true disparity d^* at (u, v) is far from the estimated disparity d, then the $c_m(u, v, d_{surr})$ will not resemble the local minima. Our membership function measures how much $c_m(u, v, d_s^*(u))$ resembles the local minima.

This computation is significantly faster than the full dense map computation approach performed in [32]. All pixels below the estimated u-v boundary, $v^*_{bottom}(u)$ computed in §3.3, do not need to be computed. If the expected maximum height of an object is predefined, all pixels above the height can also be skipped.

Our membership function $m(u, v)$ is defined as

$$m(u, v) = 2 \cdot (\max(0, m_1(u, v)) - 0.5) \tag{3.8}$$

$$m_1(u, v) = \sum_{d \in N(d_s^*(u))} \frac{m_2(\widetilde{c_m}(u, v, d), \widetilde{c_m^*}(u, v, d_s^*(u))}{|N(d_s^*(u)|} \tag{3.9}$$

$$m_2(c_m, c_m^*) = \begin{cases} +\max(|c_m - c_m^*|, \Delta_{max})/\Delta_{max} \text{ if } c_m > c_m^* \\ -\max(|c_m - c_m^*|, \Delta_{max})/\Delta_{max} \text{ otherwise} \end{cases} \tag{3.10}$$

where $\widetilde{c_m}(u, v, d)$ is the cost value after applying a mean filter. $N(d_s^*(u))$ is a small neighborhood of pixels around $d_s^*(u)$, $|N(d_s^*(u)|$ indicates the number of elements in $N(d_s^*(u))$, and Δ_{max} is a small constant. As defined in Equation 3.8, $m(u, v) \in [-1, \infty]$ where a

membership value of 1 indicates the full membership that belongs to the Stixel while a membership value of -1 means no membership.

3.4.1 Height Cost

Based on the membership function $m(u, v)$, the proposed Stixel estimator converts it into the height cost $c_h(u, v)$. We also use the expected height of the object h_o to reduce the computation time.

$$c_h(u,v) = \Big(\sum_{w=v^*_{bottom}}^{v} |m(u,w) - 1| \Big) + \Big(\sum_{w=v}^{v(h_o, d^*_s(u))} |m(u,w) + 1| \Big) \tag{3.11}$$

where $v(h_o, d^*_s(u))$ indicates the row of the maximum height considered for an object. If not defined, $v(h_o, d^*_s(u)$ should be set to 0.

3.4.2 Smoothness Term

Similar to §3.3.2, we apply a smoothness term to penalize jumps in the vertical direction.

$$s_h(u_a, v_a, u_b, v_b) = |v_a - v_b| \cdot \max(0, 1 - \frac{|z(d^*_s(u_a)) - z(d^*_s(u_b))|}{\Delta z_2}) \tag{3.12}$$

where u_a and u_b are neighboring pixels,$|u_a - u_b| = 1$. The cost of the jump is proportional to the difference between the rows v_a and v_b. Δz_2 is the minimum distance of adjacent Stixels that influence each other (set to 2 meters in our experiments). Hence, the spatial cost of a jump in the vertical direction becomes zero if the difference in depth between the columns is equal or larger than Δz_2. $z(d^*_s(u))$ denotes a depth of the optimal disparity value at column u based on the camera calibration matrix.

3.4.3 Dynamic Programming

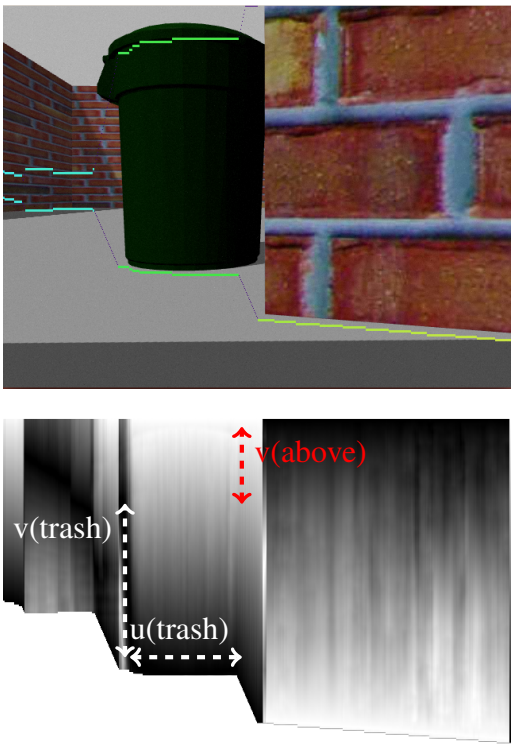
With computed height cost c_h and smoothness term s_h, the Stixel estimator also finds the optimal Stixel height, $v^*_s(u)$, by solving the 2D dynamic programming for the cost mini-

mization similar to §3.3.3 but with different data and smoothness terms.

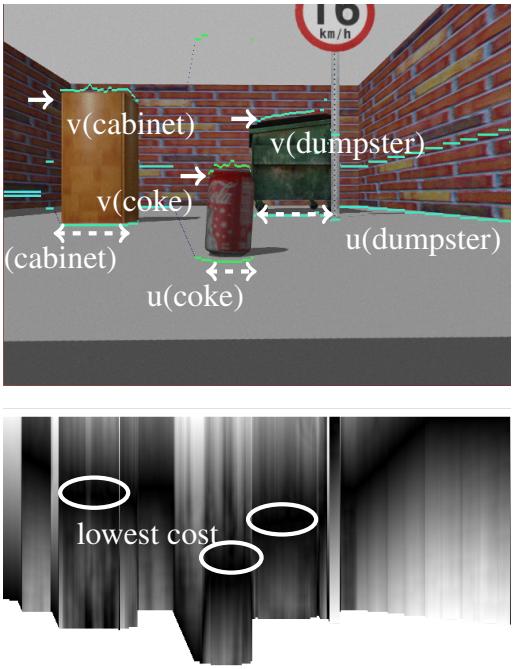
$$\begin{aligned} v_s^*(u_{width}) &= \arg\min_v c_h(u_{width}, v)) \\ v_s^*(u-1) &= \arg\min_v \{c_h(u-1, v) + s_h(u, v_s^*(u), u-1, v)\} \end{aligned} \tag{3.13}$$

As a post-processing step, if the estimated Stixel height is too far from the expected height h_o, we consider it erroneous and set back to h_o. In the height cost matrix from Figure 3.5a, the matrix shows the height cost of the region where the trash can is located in the image spcae, $c_h(u(trash), v(trash))$, has relatively smaller than $c_h(u(trash), v(above))$ where $v(above)$ is rows higher than the top of the trash can. One problem is that the height cost matrix is distributed evenly over the trash can despite the fact that the cost at the upper boundary of the object should be the lowest. This is because there is not much texture difference over the region.

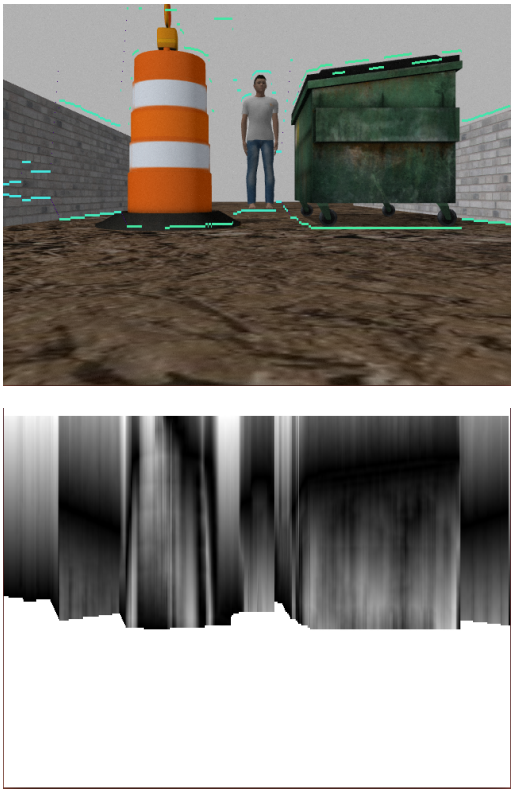
For local scenes that objects have more textures such as Figure 3.5b, we observe that the Stixel height estimation performs better by correctly estimating the upper boundary of the object with the lowest cost. From the height cost matrix, we see that the height cost $c_h(u(cabinet), v(cabinet))$, $c_h(u(coke), v(coke))$, and $c_h(u(dumpster), v(dumpster))$ are the lowest among other v for each $u(cabinet)$, $u(coke)$, and $u(dumpster)$.



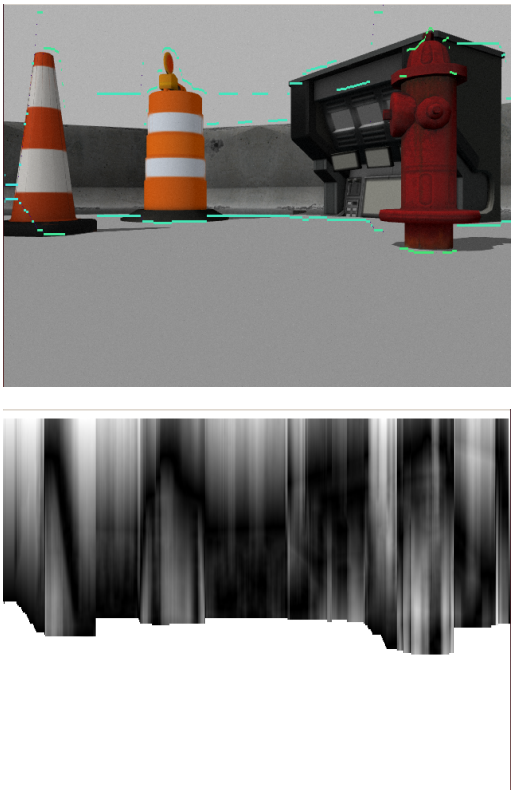
(a) Scene A



(b) Scene B



(c) Scene C



(d) Scene D

Figure 3.5: Stixel height estimation in the left image with corresponding height cost matrix

CHAPTER 4
LOCAL PATH PLANNING

During a navigation task, a local reactive controller needs to know locations of obstacles that are present in the representation that the controller performs the path planning. For the most planner that uses different types of sensors such as a laser scanner as well as some vision-based navigation planners, they usually convert and update the observation in the 2D/3D world representation and construct the path that the robot should follow within the next control stage. For vision-based planners that plan the path in the image representation, popular modality is to expand the obstacles along the boundary in the image space and construct the local plan pixel by pixel. In this paper, we propose the model projection based local path planning approach in the Stixel representation that only constructs a set of achievable trajectory candidates based on the robot's kinematics and dynamics at every control stage.

4.1 Model Projection Based Path Planning

Traditional collision checking approaches expand either a robot or an obstacle boundary by simplifying the geometrical shape of the object, which usually over-expands the object so that the resulting plan becomes overly permissive.

As the gap between the geometrical shape of the actual robot and the expansion model in the image representation gets larger, the local path planner has to choose more conservative paths in order to avoid the collision. This deficiency sometimes affects the overall navigation performance in the cluttered environments by not only causing a long detour but also not being able to find a valid path frequently.

Instead, the work by Smith and Vela [28] presents a local path planning approach that models the 3D physical geometry of the robot and projects it directly into the perception

space to find collision-free trajectories. The advantage of this approach is that the collision checking is done in a 3D volume but only requires 2D image comparisons. Another advantage is that the approach does not need to maintain the world representation unlike most vision-based path planning algorithms that update the observation. With more precise trajectory evaluation in the perception space compared to obstacle expansion based approaches, the navigation system can construct more accurate local plans during the navigation task.

In this paper, we propose a similar collision checking algorithm for local/reactive scene analysis and path planning that uses Stixel disparity instead of depth image. This chapter assumes that there is a set of trajectories to evaluate provided by a global planner based on the status of the robot. Hence, the local path planner does not check the validity of the trajectory and assume that the trajectory is feasible for the robot to achieve within the next control stage. How the global planner constructs and validates each trajectory will be discussed in the next chapter. For convention, we use a camera optical frame axis with x-axis right, y-axis down, and z-axis forward.

4.2 3D Cartesian Point Projection

Since the proposed collision checking method operates in the image representation, we first briefly revisit the image coordinate projection of a 3D Cartesian point. From a pinhole camera model, the mapping between the 3D Cartesian point (x, y, z) and an image coordinate

(u, v) on the left image of the stereo camera is the following:

$$
\begin{aligned}
Z_l \begin{bmatrix} u \\ v \\ l \end{bmatrix} &= K \, [I|0] \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\end{aligned}
\tag{4.1}
$$

where f_x and f_y are the focal lengths of the camera and $[c_x, c_y]$ is the principal point of the camera optical axis in pixels. R and T are the rotation and translation matrices from the world frame that the 3D Cartesian point is constructed to the left camera frame. If the 3D point is constructed in the left camera frame, R and T matrix should be identity and zero matrices respectively.

4.3 Implementation of Robot Model Projection

In this paper, we use a Turtlebot 2 as our navigation robot as shown in Figure 4.1 and implement it as our robot model in the perception space.

Figure 4.1: Robot platform used for implementation and experiments

Based on the actual robot, we implement two robot models in the perception space: Rectangular and Cylindrical model. The rectangular model projection assumes a frontal side of the robot as a rectangle while the cylindrical model tries to represent the actual robot model more closely. A trade-off between these two models is the time complexity and quality of trajectory evaluation. The local path planner is able to create the rectangular robot model in the perception space significantly faster than the cylindrical model but still simplifies the 3D geometry of the actual robot, which eventually addresses the overly conservative navigation as well. The cylindrical model reflects the actual robot much more closely and yields more precise trajectory evaluation. However, it also requires more computation to construct.

4.3.1 Rectangular Model

For a projection of the rectangular model at a 3D Cartesian point (x, y, z), the planner first calculates the 3D Cartesian positions of the rectangular corners that bounds the frontal side

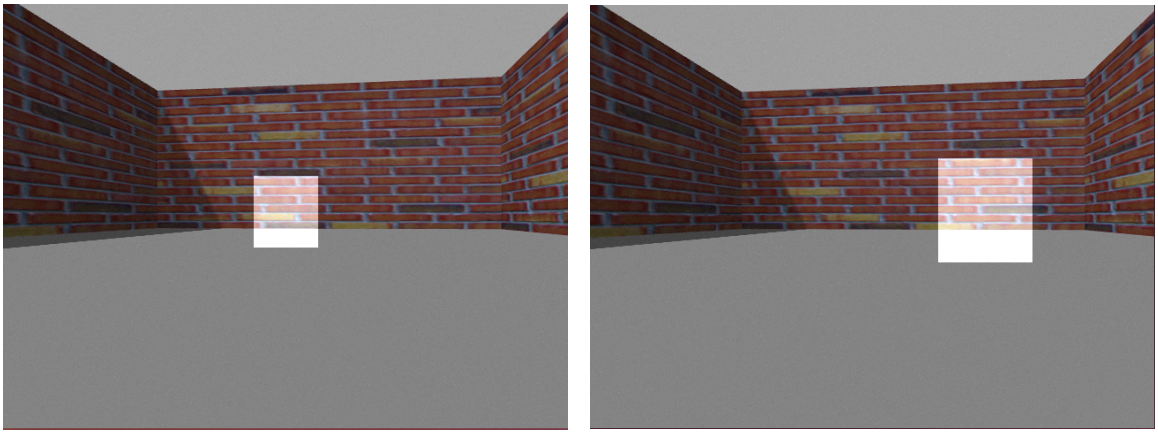
of the robot.

$$
\begin{aligned}
top_l &= (x - r - e_s,\ y - h,\ z + r + e_s) \\
top_r &= (x + r + e_s,\ y - h,\ z + r + e_s) \\
bottom_l &= (x - r - e_s,\ y,\ z + r + e_s) \\
bottom_r &= (x + r + e_s,\ y,\ z + r + e_s)
\end{aligned}
\tag{4.2}
$$

where r, h, e_s are a radius of the base, height of the robot, and safety expansion of the projected robot model respectively. Then the planner projects each 3D Cartesian point in the image space using the equation described in §4.2.

From images in Figure 4.2, we show projected rectangular models at 3D Cartesian points. In the image, the area colored white is a rectangular model that bounds the frontal side of the robot. We briefly describe the downside of using the rectangular model in the perception space. From the rectangular model projection shown in Figure 4.2a and 4.2b, the projected robot model in the image space is a rectangular shape while the actual robot has a circular front side. This is a case of the over-expansion on the frontal side, and the navigation planner will discard a trajectory if the over-expanded area overlaps with an obstacle, which causes the conservative path selecting problem.

Hence, the local planner will discard a path that the far left or right side of the projected model overlaps with the obstacle even though it is traversable in reality. Another downside is that the projected model has a constant depth over a column. Based on the geometry of the actual robot shown in Figure 4.1, each column should have different depth while the center of the model has the largest depth. Since our rectangular model projection method assigns the depth of the farthest point from the center to all the columns as expressed in Equation 4.2, the collision checking by comparing the depth in the model becomes simplified.



(a) Projection of a point (3m forward)

(b) Projection of a point (2m forward, 0.5m to the right

Figure 4.2: Rectangular Model Projection

4.3.2 Cylindrical Model

For a projection of the cylindrical robot model at a 3D Cartesian point (x, y, z), we first compute the far left and right side of the projected model. Denoted as points L and R in Figure 4.3, these points are the far left and right boundary of the projected model that the pinhole camera is able to see in the image space. We compute these points of tangency between the current camera optical origin and the projected robot model using the following equations:

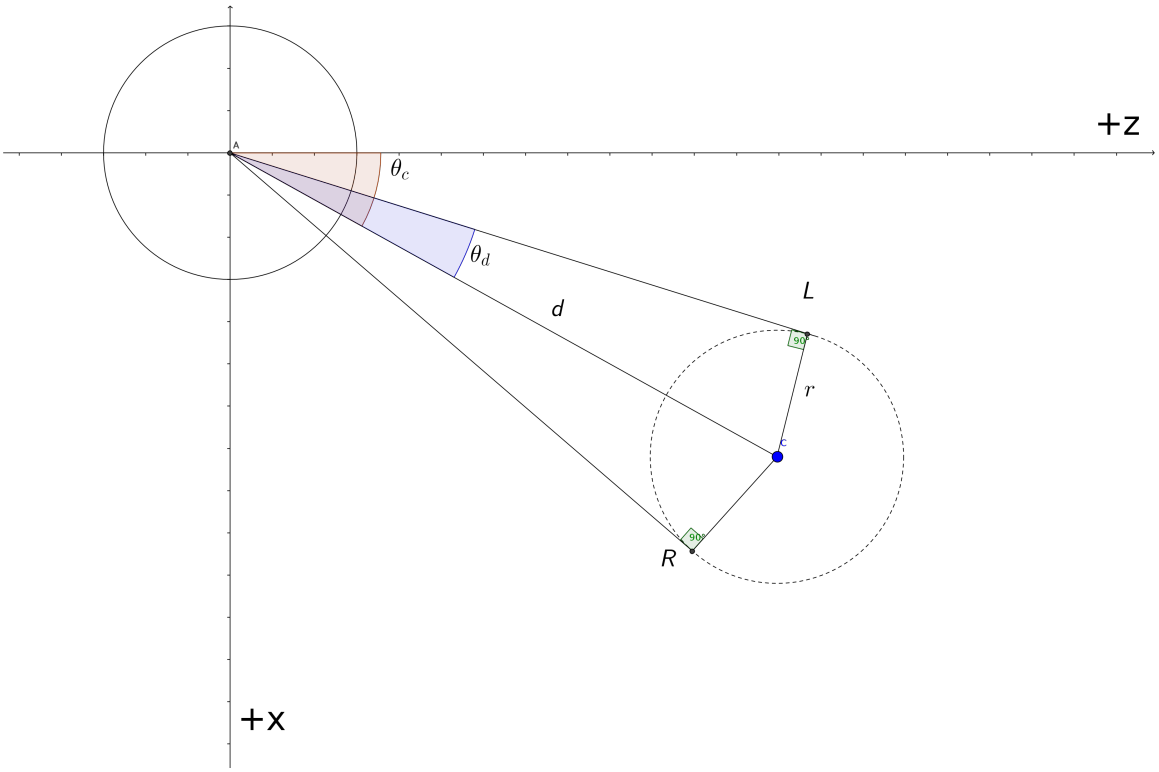


Figure 4.3: Cylindrical model projection in 2D Cartesian space

$$
\begin{aligned}
bottom_l &= (d_t \sin(\theta_c - \theta_d),\ y,\ d_t \cos(\theta_c - \theta_d)) \\
bottom_r &= (d_t \sin(\theta_c + \theta_d),\ y,\ d_t \cos(\theta_c + \theta_d))
\end{aligned}
\tag{4.3}
$$

$$
\begin{aligned}
d_t &= \sqrt{d^2 - r^2} \\
\theta_c &= \tan^{-1} \frac{x}{z} \\
\theta_d &= \sin^{-1} \frac{r}{\sqrt{x^2 + z^2}}
\end{aligned}
\tag{4.4}
$$

where $bottom_l$ and $bottom_r$ are the bottom left and right points in the 3D Cartesian space.

Then, the planner projects $bottom_l$ and $bottom_r$ in the image space using §4.2 to get far left and right coordinates of the robot model in the perception space. Based on the actual robot shown in Figure 4.1, the y coordinates of the projected model in the Cartesian space are constant along the boundary of the robot base while x and z coordinates (x-axis right, y-axis down, and z-axis forward) change along each column. Hence, the planner finds

x and z coordinates by computing an intersection of the projected circle and the vector that comprises of a unit vector and distance. The computation of the 3D Cartesian point (x_b, y_b, z_b) that represents the bottom of the robot base at column u is the following:

$$\begin{aligned} x_b(u) &= \gamma_x \cdot t \\ y_b(u) &= \gamma_y \cdot t = y \\ z_b(u) &= \gamma_z \cdot t \end{aligned} \tag{4.5}$$

where γ and t are the computed unit vector and the scalar distance from the camera origin to the Cartesian point.

Based on the property of a circle, we construct a center-radius form of the circle equation.

$$(x_b - x)^2 + (z_b - z)^2 = r^2 \tag{4.6}$$

Using the quadratic equation, t can be solved as follows:

$$t = \frac{-(-2x\gamma_x - 2z\gamma_z) + \sqrt{(-2x\gamma_x - 2z\gamma_z)^2 - 4(\gamma_x^2 + \gamma_z^2)(x^2 + z^2 - r^2)}}{2(\gamma_x^2 + \gamma_z^2)} \tag{4.7}$$

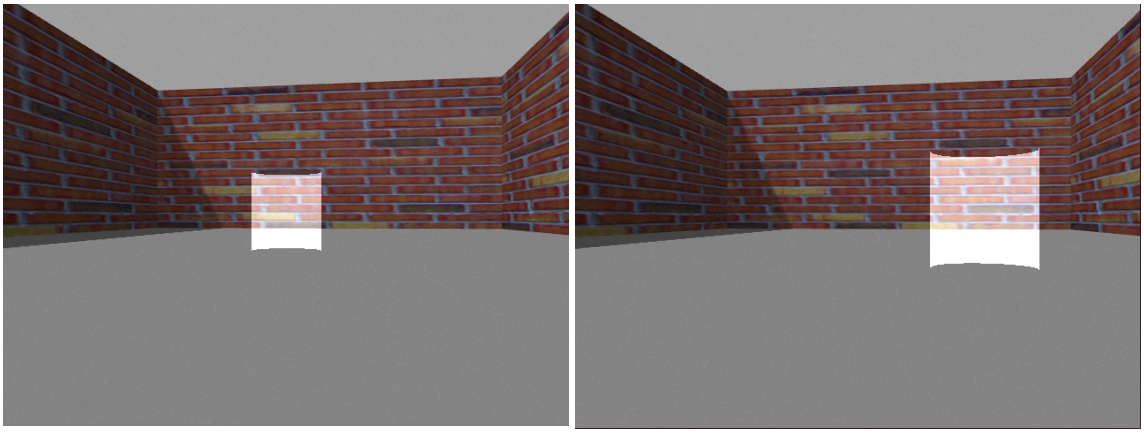
A derivation of Equation 4.7 can be found in Appendix B. Finally, bottom and top 3D Cartesian coordinates for each column u can be expressed as follows:

$$\begin{aligned} bottom(u) &= (x_b(u),\ y,\ z_b(u)) \\ top(u) &= (x_b(u),\ y - h,\ z_b(u)) \end{aligned} \tag{4.8}$$

The planner then projects these 3D Cartesian points back into the image space using the equation described in §4.2.

Hence, the vertical start and end points for each column is different as shown in Figure 4.4a and 4.4b based on the different depth. Using the cylindrical model, the projected model reflects the actual robot much closer than the rectangular model projection so that

the local path planner can perform collision checking of the trajectory more precisely.



(a) Projection of a point (3m forward)

(b) Projection of a point (2m forward, 0.5m to right)

Figure 4.4: Cylindrical Model Projection

4.4 Collision Checking

We propose a collision checking algorithm based on the Stixel disparity and the projected robot model. The proposed algorithm assumes that a trajectory candidate, T_i, consists of poses sampled in time and checks collision by comparing the Stixel disparity and depth of the projected robot model at each pose.

For a pose $p_{t_i}(x, y, z) \in T_i$ the planner constructs the projected robot model as described in §4.1. When the planner constructs the corresponding robot model at the pose $p(x, y, z)$ in the image representation, the collision checking algorithm evaluates that the robot pose at the point is in collision with an object in the real world if the estimated depth from the Stixels is less than that of the projected robot model in the image. Hence, a pose is collision-free if

$$Collision_{free}(x, y, z) = \begin{cases} \text{true}, & \text{if} D^*_m(u) > D^*(u)\ \forall u \in [u_l, u_r] \\ \text{false}, & \text{otherwise} \end{cases} \tag{4.9}$$

where, u_l and u_r are far left and right horizontal image coordinates of the robot model

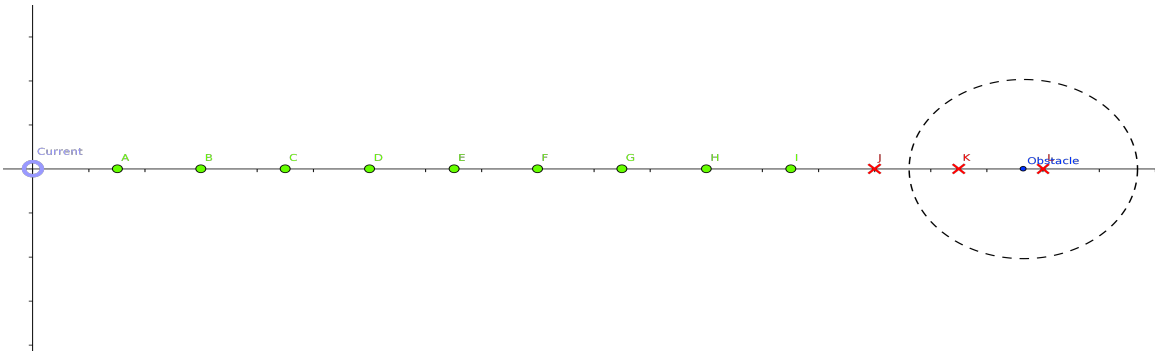
projected at point (x, y, z).

Since the depth is deterministic from the estimated disparity, we compute the depth of the Stixel at each u.

$$D^*(u) = \frac{fB}{d^*(u)} \tag{4.10}$$

where f and B are the focal length and baseline of the camera respectively.

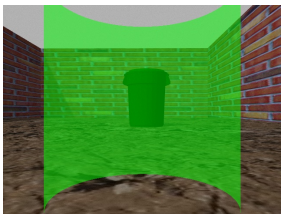
In both Figure 4.5 and 4.6, the planner evaluates the point J as a collision point. The robot model is projected at point J and finds the collision between the frontal side of the robot and a trash can by comparing the depth of each column of the projected model and the depth of the Stixels that correspond to the column.



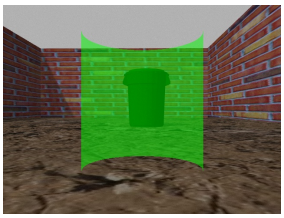
(a) Sample Trajectory A in 2D Cartesian grid, collision points are denoted as "X"

(b) Actual robot placement on each pose in Trajectory A

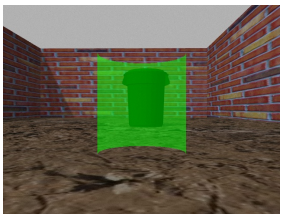
(c) Point A



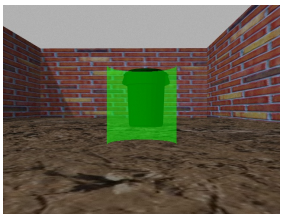
(d) Point B



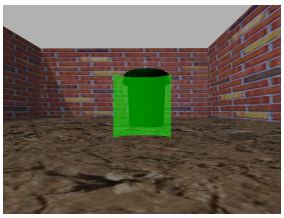
(e) Point C



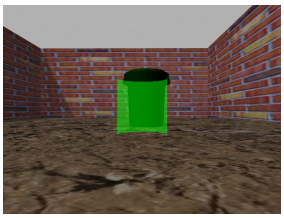
(f) Point D



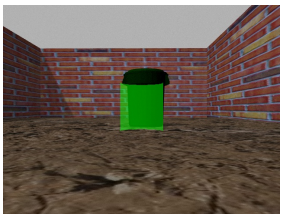
(g) Point E



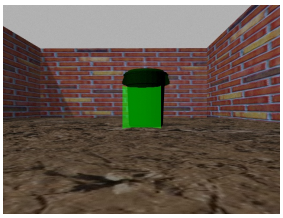
(h) Point F



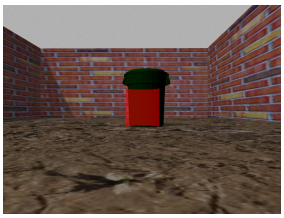
(i) Point G



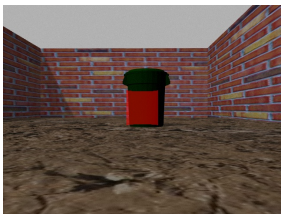
(j) Point H



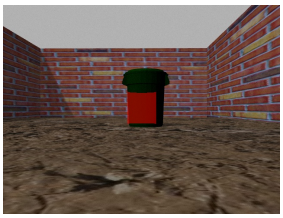
(k) Point I



(l) Point J

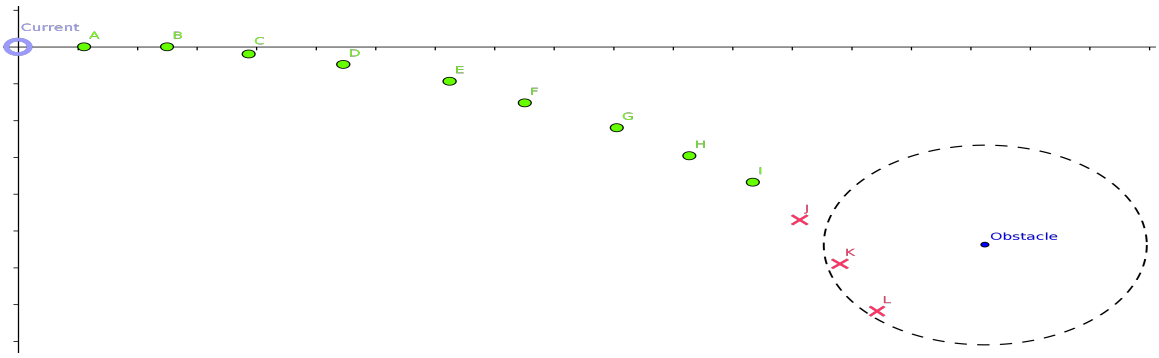


(m) Point K

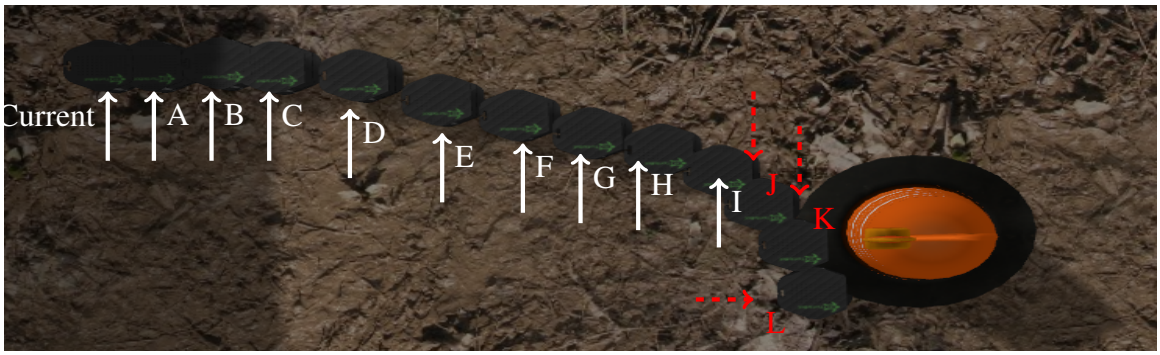


(n) Point L

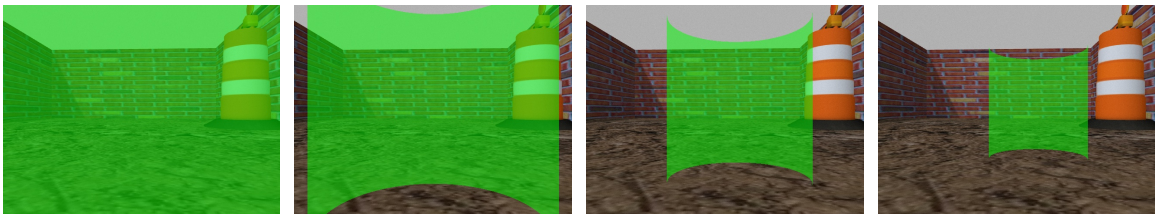
Figure 4.5: Collision Checking in the image space at point A - L



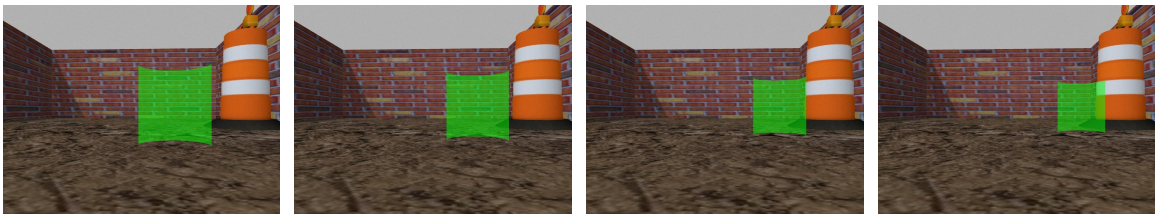
(a) Sample Trajectory B in 2D Cartesian grid, collision points are denoted as "X"



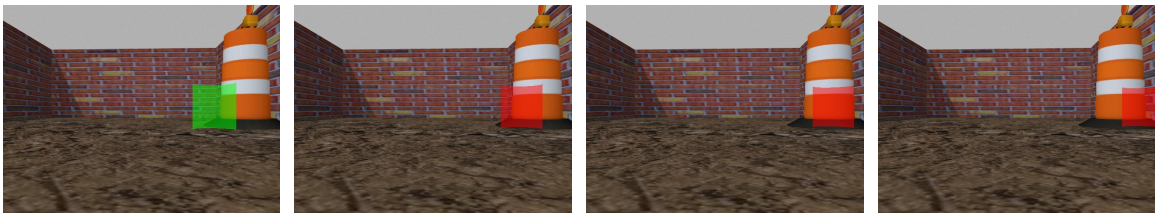
(b) Actual robot placement on each pose in Trajectory B



(c) Point A (d) Point B (e) Point C (f) Point D



(g) Point E (h) Point F (i) Point G (j) Point H



(k) Point I (l) Point J (m) Point K (n) Point L

Figure 4.6: Collision Checking for Trajectory B in the image space at point A - L

4.5 Local Reactive Controller

Before integrating our approach with the global navigation framework, we first implement our collision checking algorithm as a local reactive controller that navigates thorough obstacles without any goal. Hence, the mobile robot moves through obstacles until a collision occurs.

For the implementation, we use a trajectory library that generates trajectories simulated in time based on a departure angle from the current position of the robot. Using the trajectory generation method of [28], the controller constructs a set of trajectories as follows.

$$T_i^*(t) = v_{nom} R(\theta_{dep,i}) e_1 t \text{ for } t \in [0, t_{max}] \tag{4.11}$$

where v_{nom} is a desired forward velocity, $R(\theta)$ is a rotation matrix that rotates the frame by θ, e_1 represents a unit vector in the body x-direction in the robot base frame (*i.e.*, forward for the robot), and t_{max} is the duration of the trajectory.

Applying the controller to each straight trajectories creates an indexed set of achievable trajectories $T = (g_i^*(t), V_i^*(t))$ that satisfy kinematic and dynamic constraints of the robot. Since only purpose of the local reactive controller is to navigate without collision, the planner selects the trajectory that maximizes the safe travel time,

$$\underset{(g(t),V(t))\in\chi}{\arg\max} \; T_{safe}(g(t), V(t)) \tag{4.12}$$

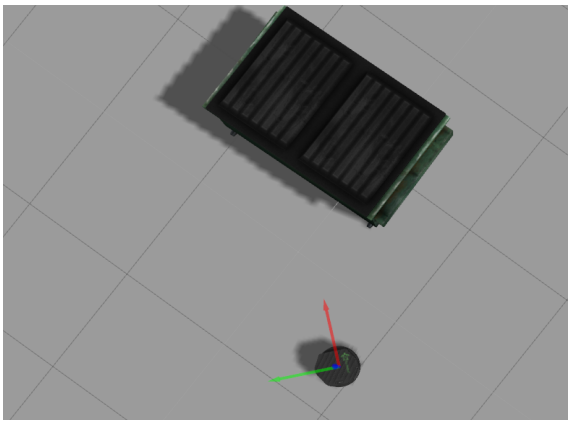
To find the safe travel duration of the trajectory (up to t_{max}), we apply the Stixel collision testing procedure from §4.4:

$$T_{safe}(g(t), V(t)) = \underset{t\in[0,t_{max}]}{\arg\max} \, Collision_{free}(pos \circ g(t)) \tag{4.13}$$

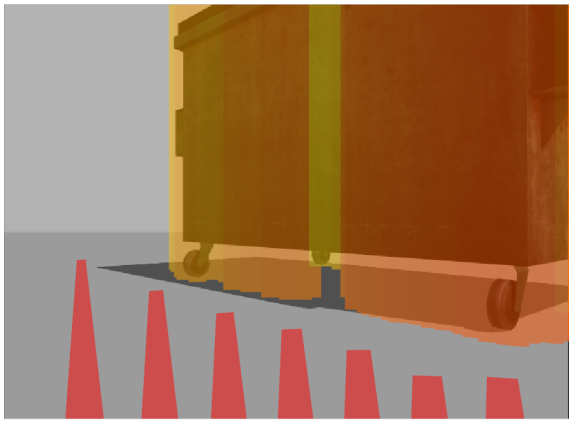
where pos is the translational position of $g \in SE(2)$.

We show the evaluation of trajectory candidates using our approach in different local

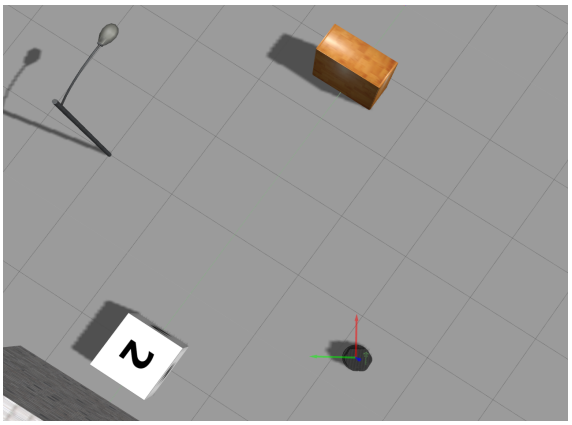
scenes in Figure 4.7. In these images, Stixels are colored based on the Stixel disparity. From Figure 4.7b, the robot looks at a dumpster and treats it as an obstacle based on the Stixel disparity estimation. Hence, the trajectories that goes to the dumpster have shorter T_{safe}. In contrast, T_{safe} for all of the trajectories in Figure 4.7d are fairly similar because the location of the cabinet is farther than $g(t_{max})$. These two cases occur frequently during local reactive navigation tasks. However, there can be occasions that the robot is trapped in the cluttered environment due to the limitations of the camera similar to Figure 4.7f. This is the case where the local reactive controller is unable to find a safe trajectory due to the small field of view and nearby obstacles. As a result, the local reactive controller is unable to navigate this type of local scenes since it does not maintain knowledge of obstacle position nor any alternative paths that it can travel. Moreover, the trajectory generation method creates a fixed set of trajectories so that the planner is unable to navigate due to the coarseness of the path samples. Therefore, we discover that the planner needs to construct trajectories with various velocity samples. From the results, we show that the local planner alone is insufficient for an end-to-end global navigation system.



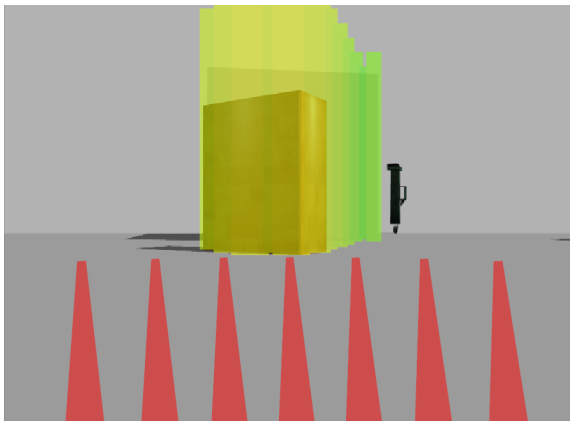
(a) World view of Scene A



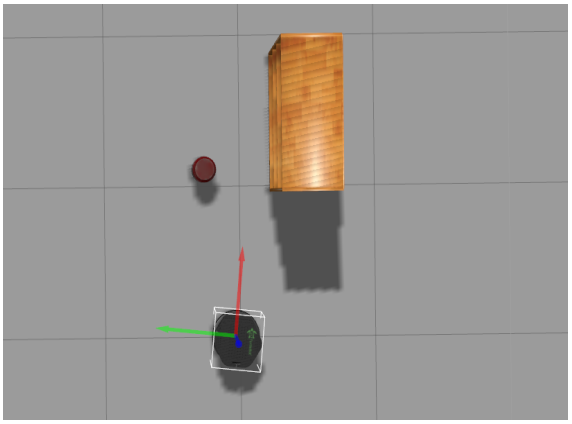
(b) Analysis of Scene A in perception space



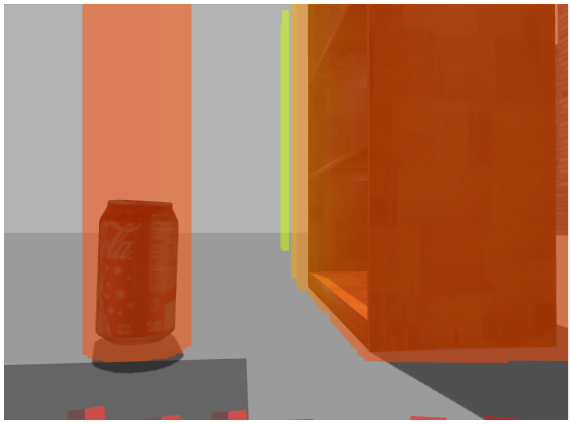
(c) World view of Scene B



(d) Analysis of Scene B in perception space



(e) World view of Scene C



(f) Analysis of Scene C in perception space

Figure 4.7: Example path evaluation in different local scenes

CHAPTER 5
GLOBAL NAVIGATION FRAMEWORK

To function effectively as an end-to-end navigation framework, the local reactive approach in §4.5 must integrate with additional features and capabilities. A core idea is to transform the local navigation process into serialized goal-driven sub-processes, where each sub-process considers the location of the destination. In this paper, we assume that the navigation planner knows the current position and state of the robot and the goal. Again, the navigation planner does not interpret and convert information from the perception space into the world representation; it only tracks the current location and destination in the world coordinate system.

5.1 Global and Local Plan

A global plan is a high-level path that the local path planner uses as a reference during a navigation task. The global path planning requires the environment to be completely known and the terrain should be static. During the global path planning stage, the path planner generates a complete path from the start point to the destination before the robot starts its motion. Therefore, the global plan does not consider the kinematic and dynamic constraints of the robot that dynamically change while navigating.

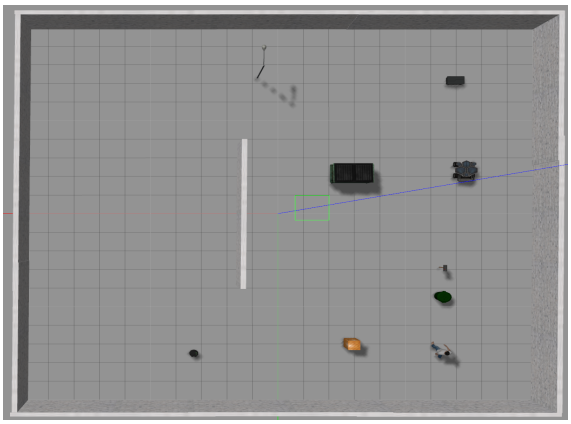
On the other hand, a local plan is a reactive plan that the robot is going to follow at every control loop. Hence, the planned local path should be feasible and reachable based on the hardware limitations of the robot in a given period. In other words, the local planner should be capable of generating a new plan in response to environmental changes. In most cases, the local planner constructs the plan from the current position of the robot with an intermediate destination called a local goal. In our implementation, we select the local goal based on the effective sensing range of the sensor so that a constructed set of trajectory

candidates can be evaluated in the given range. Hence, we implement a rolling window approach which the size of window is established based on the sensing range of the stereo camera. In the paper, we use a size of 6x6 meters of a rectangular box and select an intersection between the window and the global plan as the local goal.

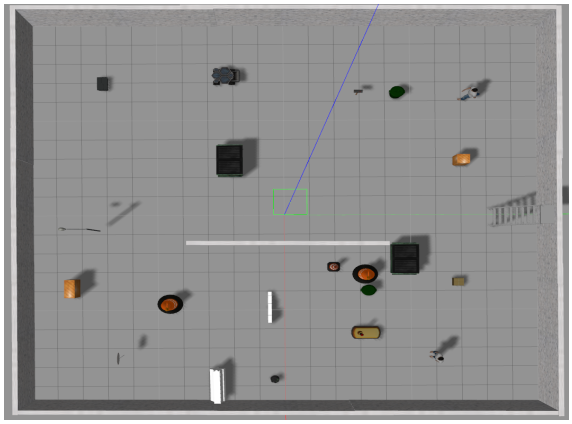
The planner can naively travel towards the local goal at every control frame so that the navigation task can be completed by following the shortest path, if there is a guarantee that all of the obstacles are known and already represented in the world. In Figure 5.1, we show constructed global plans of two local scenes in different representation spaces. For these environments, we add the large wall in the middle as a known object so that the planner can construct the global plan that avoids the known obstacle. We treat other objects such as the dumpster, cabinet, or barrels as unknown objects. We spawn the robot on the bottom of the world and provide the destination to the top of the world represented as a red arrow shown in Figure 5.1c and 5.1d. We show a case when the local planner can travel towards the local goal without scene analysis in Figure 5.1e. The local planner can directly follow the global plan since there are no unknown objects along the path. For the global plan in Figure 5.1f, however, following the global plan without constant local scene analysis causes potential collisions with obstacles. Since the latter is the most common case during the navigation task, we always assume that there will be unknown objects along the constructed global plan. Hence, the navigation planner must construct a new reactive local plan based on the environmental changes at every planning stage so that the robot avoids the obstacles and travels to the destination both safely and efficiently.

In our implementation, the planner constructs the global plan from the initial position of the robot to the destination using Dijkstra's algorithm[47]. Given a premade map, the planner divides the world into smaller cells as a grid. We use a resolution of 5cm for each cell.

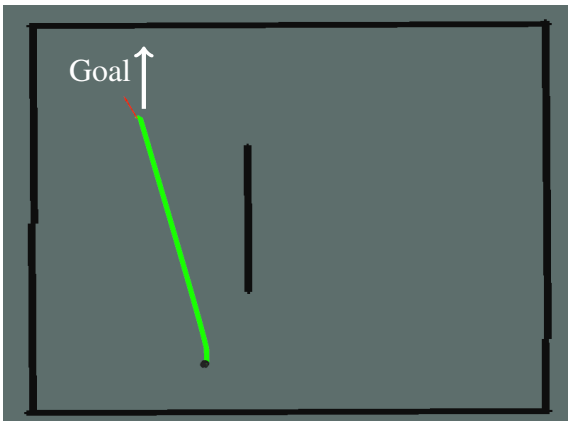
Algorithm 2 describes the Djikstra's path searching algorithm. This algorithm assumes to have varying costs of moving from one cell to another for different cells. Since the



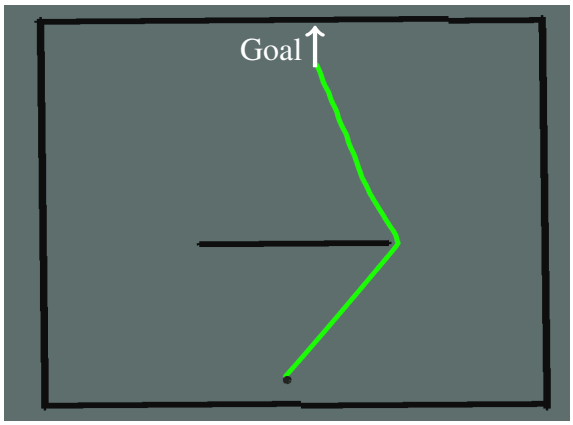
(a) 3D world view of Scene A



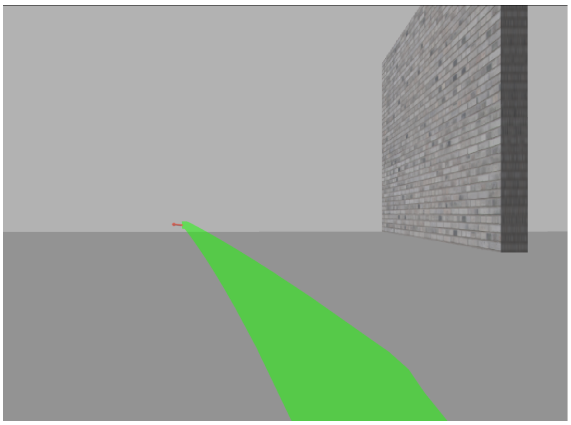
(b) 3D world view of Scene B



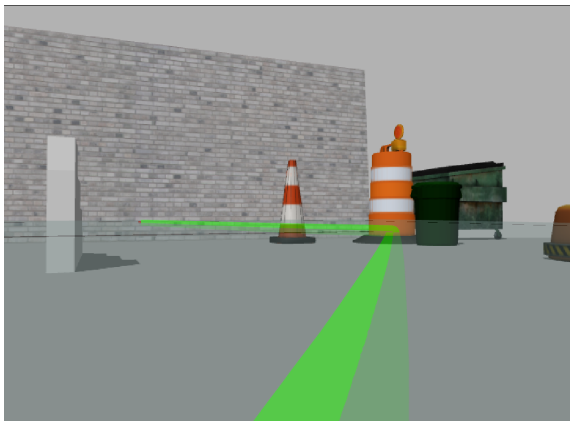
(c) 2D grid representation of Scene A with a connstructed global plan



(d) 2D grid representation of Scene B with a constructed global plan



(e) Constructed global plan in the perception space for Scene A



(f) Constrcuted global plan in the perception space for Scene B

Figure 5.1: Global plan visualization in three representations

planner does not update the grid while navigating, the path searching algorithm can apply constant values for both moving between free cells and occupied cells (*e.g.* 1 and 100000).

Algorithm 2 Djikstra-based path searching

```
1:  procedure DJIKSTRA(grid, current)
2:      for each vertex v ∈ grid do
3:          cost[v] ← ∞
4:          previous[v] ← undefined
5:      cost[current] ← 0
6:      Q ← all cells in the grid
7:      while Q is not empty do
8:          u ← vertex in Q with min cost[u]
9:          remove u from Q
10:         for each neighbor v of u do
11:             temp ← cost[u] + cost(u, v)
12:             if temp < cost[v] then
13:                 cost[v] ← temp
14:                 previous[v] ← u
```

For certain cases that the planner needs to update the obstacle information on the grid to construct an alternative global plan to exit the area (*e.g.* Figure 4.7f), the planner might need to update the grid with different costs on each cell based on the situation.

This stage does not significantly affect performance regarding the computation speed during the navigation task because construction of the global plan usually occurs only once during the initialization process or few rare cases that the local controller needs a new global plan for specific reasons (*e.g.* global replanning, exit procedure, and recovery behavior).

5.2 Sample Based Trajectory Generation

Once the local planner receives the global plan, it constructs a set of trajectory candidates in the velocity space. One important constraint in our trajectory generation approach is that a desired velocity $v(x, y, \theta) \in candidate_i$ must be reachable and feasible within the next control loop. Therefore, the generation algorithm limits the range of velocity samples based on the dynamic and kinematic constraints of the robot. Then, the planner selects the

local goal for local path planning, which must be reasonably far to travel but not so far that it goes outside of the effective depth sensing range of the camera. We first calculate the effective depth range of the stereo camera based on the configuration of the camera prior to the navigation task.

$$Z = \frac{fB}{d} \tag{5.1}$$

where f and B are focal length and baseline of the stereo camera and d can be 1 for pixelwise stereo matching.

We use a rolling rectangular window with a size of 6x6 meters based on the depth range computation. As already mentioned above, we select the local goal as the intersection point of the rolling window and the global plan.

For construction of the trajectory set, the local planner first computes the reachable velocity based on the current state of the robot.

$$\begin{aligned}
v'_{max_x} &= \min(V_{max_x}, v_x + a_{lim_x} t_s),\ v'_{min_x} = \max(V_{min_x}, v_x - a_{lim_x} t_s) \\
v'_{max_y} &= \min(V_{max_y}, v_y + a_{lim_y} t_s),\ v'_{min_y} = \max(V_{min_y}, v_y - a_{lim_y} t_s) \\
v'_{max_\theta} &= \min(V_{max_\theta}, v_\theta + a_{lim_\theta} t_s),\ v'_{min_\theta} = \max(V_{min_\theta}, v_\theta - a_{lim_\theta} t_s)
\end{aligned} \tag{5.2}$$

where v'_{max} and v'_{min} denote the upper and lower bounds of the velocity that is reachable within the next planning loop based on the current velocity $v(v_x, v_y, v_\theta)$. V_{max}, V_{min}, and a_{lim} are the maximum and minimum velocities and the acceleration limit based on the hardware of the robot. t_s denotes the control period.

Based on the computed upper and lower bound, the planner creates the velocity sample space. For a given velocity sample $(v_{target_x}, v_{target_y}, v_{target_\theta})$, the velocity that the robot can

reach at each timestep is the following:

$$\begin{bmatrix} v'_x(n) \\ v'_y(n) \\ v'_\theta(n) \end{bmatrix} = \begin{bmatrix} \begin{cases} \min(v_{target_x},\ v'_x(n-1) + a_{lim_x}\Delta t), \textit{if } v_{target_x} > v'_x(n-1) \\ \max(v_{target_x},\ v'_x(n-1) - a_{lim_x}\Delta t), \textit{otherwise} \end{cases} \\ \begin{cases} \min(v_{target_y},\ v'_y(n-1) + a_{lim_y}\Delta t), \textit{if } v_{target_y} > v'_y(n-1) \\ \max(v_{target_y},\ v'_y(n-1) - a_{lim_y}\Delta t), \textit{otherwise} \end{cases} \\ \begin{cases} \min(v_{target_\theta},\ v'_\theta(n-1) + a_{lim_\theta}\Delta t), \textit{if } v_{target_\theta} > v'_\theta(n-1) \\ \max(v_{target_\theta},\ v'_\theta(n-1) - a_{lim_\theta}\Delta t), \textit{otherwise} \end{cases} \end{bmatrix} \tag{5.3}$$

and

$$\begin{bmatrix} v'_x(0) \\ v'_y(0) \\ v'_\theta(0) \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_\theta \end{bmatrix} \tag{5.4}$$

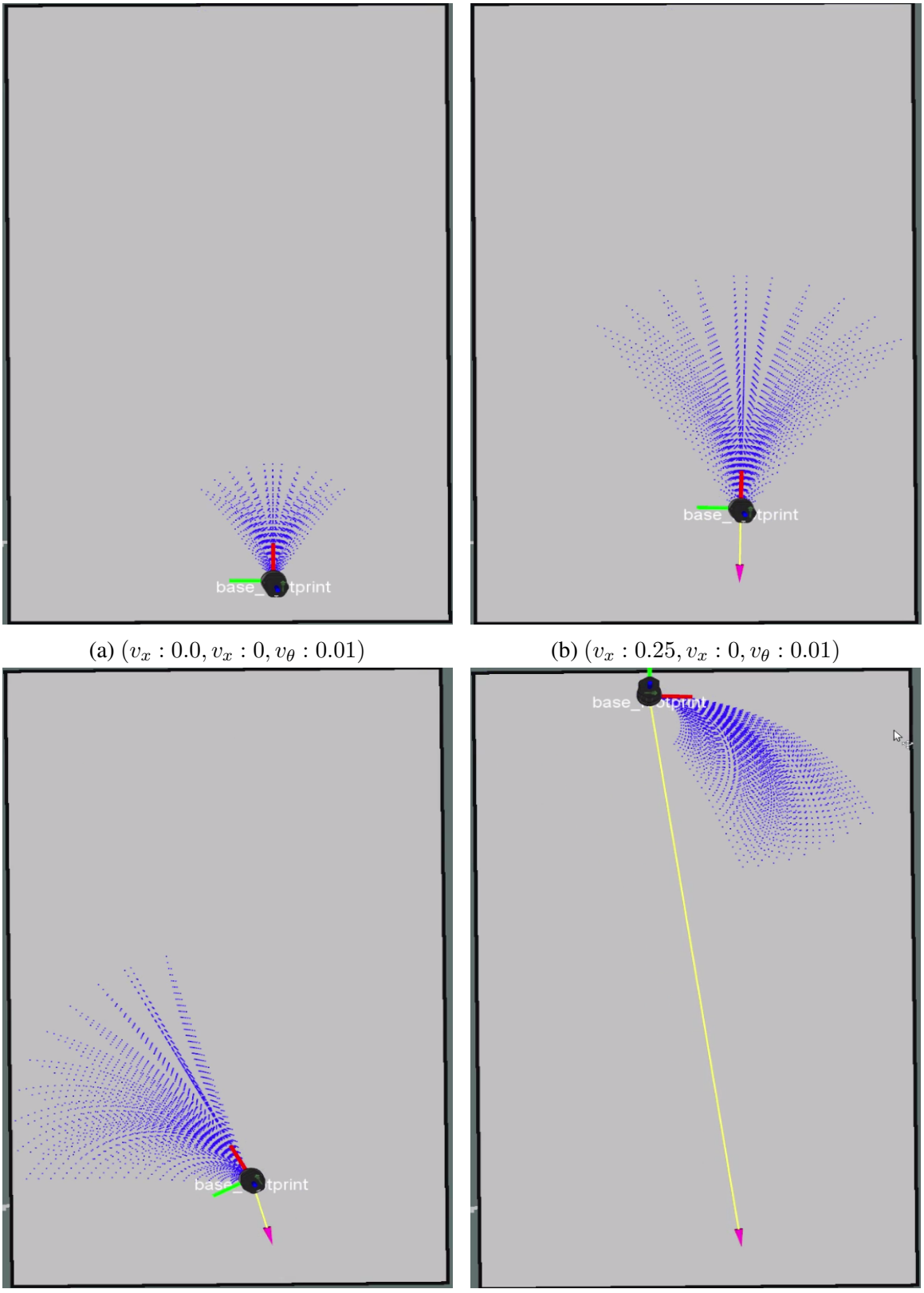
where Δt is t_s divided by a number of poses that the planner is set to sample for each candidate. $v'(n)$ and v denote the sampled velocity at timestep n and the current velocity of the robot. Using Equation 5.3, the planner then calculates the 2D Cartesian points for the non-holonomic robot at each timestep.

$$\begin{bmatrix} pos_x(n) \\ pos_y(n) \\ \theta(n) \end{bmatrix} = \begin{bmatrix} pos_x(n-1) + v_x(n-1)\cos(\theta(n-1))\Delta t \\ pos_y(n-1) + v_x(n-1)\sin(\theta(n-1))\Delta t \\ \theta(n-1) + v_\theta(n-1)\Delta t \end{bmatrix} \tag{5.5}$$

where

$$\begin{aligned} pos_x(0) &= pos_{current_x} \\ pos_y(0) &= pos_{current_y} \\ \theta(0) &= current_\theta \end{aligned} \tag{5.6}$$

In Figure 5.2, we show that various initial velocities have different reachable velocity space which creates distinctive trajectory candidates. For all the sampled trajectory candidates in the figure, we apply the same simulation time (t_s), number of samples for each trajectory candidate, maximum and minimum velocity limits (V_{max}, V_{min}), and acceleration limit a_{lim}. The average travel distance of trajectory candidates in Figure 5.2a is generally shorter than 5.2b because of its slower initial velocity. With a larger magnitude of v_θ, trajectory candidates in Figure 5.2d are more biased to the direction of the robot's current v_θ than Figure 5.2c.



(a) $(v_x : 0.0, v_x : 0, v_\theta : 0.01)$

(b) $(v_x : 0.25, v_x : 0, v_\theta : 0.01)$

(c) $(v_x : 0.25, v_x : 0, v_\theta : 0.12)$

(d) $(v_x : 0.39, v_x : 0, v_\theta : -0.31)$

Figure 5.2: Trajectory candidates sampled based on the current configuration of robot

5.3 Trajectory Evaluation

The path planner evaluates each trajectory candidate constructed in §5.2 using an objective function. As opposed to the local reactive controller described in §4.5, the objective function should not only minimize collision probability but also select the trajectory that leads the robot to the destination as fast as possible. Our objective function contains six cost functions.

- Oscillation Cost
- Local Goal Heading Cost
- Global Goal Heading Cost
- Local Goal Distance Cost
- Global Goal Distance Cost
- Obstacle Cost

In our implementation, the planner discards a trajectory if any of the cost function returns a negative value.

5.3.1 Oscillation Cost

An oscillation cost prevents the robot from moving in different directions at every control stage. An important idea behind the oscillation cost is that a trajectory candidate should be discarded if it drives the robot into the opposite direction with a magnitude greater than a predefined tolerance.

Therefore, the oscillation cost establishes two parameters called d_{trans} and θ_{rot} which are translational and rotational tolerances that can be applied when the candidate trajectory drives into the opposite direction from the current heading of the robot. The mathematical

definition is the following:

$$c_{oscillation} = \begin{cases} 0, \textit{if } (O_x \wedge O_y \wedge O_\theta) \\ -1, \textit{otherwise} \end{cases} \tag{5.7}$$

$$\begin{aligned} O_x &= \begin{cases} \textit{true, if}(v_x \cdot v'_x > 0 \vee d < d_{trans}) \\ \textit{false, otherwise} \end{cases} \\ O_y &= \begin{cases} \textit{true, if}(v_y \cdot v'_y > 0 \vee d < d_{trans}) \\ \textit{false, otherwise} \end{cases} \\ O_\theta &= \begin{cases} \textit{true, if}(v_\theta \cdot v'_\theta > 0 \vee \theta < \theta_{rot}) \\ \textit{false, otherwise} \end{cases} \end{aligned} \tag{5.8}$$

where v is the current velocity and v' is the velocity target of the trajectory candidate. d and θ are the difference in the position and angle between the current state of the robot and the first few expected robot state from the candidate trajectory. If there is no oscillation in motion, the constant cost of 0 is assigned to the trajectory.

5.3.2 Local Goal Heading Cost

During the navigation task, the planner needs to prefer a trajectory that drives the robot nose (heading) towards the local goal. Based on the local goal chosen in §5.2, the planner measures how closely the end pose of the trajectory candidate will point towards the local goal. The local goal heading cost $c_{\theta_{local}}$ is the following:

$$\begin{aligned} c_{\theta_{local}} &= |\theta_{local} - \theta| \\ \theta_{local} &= \tan^{-1}(\frac{pos_{local_y} - pos_{current_y}}{pos_{local_x} - pos_{current_x}}) \\ \theta &= \tan^{-1}(\frac{pos_{traj_y} - pos_{current_y}}{pos_{traj_x} - pos_{current_x}}) \end{aligned} \tag{5.9}$$

where pos_{traj_x} and pos_{traj_y} denote x and y coordinates of the end position in the trajectory candidate. pos_{local} and $pos_{current}$ are the position of the local goal and the robot.

5.3.3 Global Goal Heading Cost

The navigation planner should not get stuck in the local minima problem where the planner always tries to reach the local goal. Therefore, we implement a global goal heading cost that balances with the local goal heading cost. This cost measures how closely the end pose of the trajectory candidate will point towards the global goal.

$$\begin{aligned} c_{\theta_{global}} &= |\theta_{global} - \theta| \\ \theta_{global} &= \tan^{-1}\left(\frac{pos_{global_y} - pos_{current_y}}{pos_{global_x} - pos_{current_x}}\right) \\ \theta &= \tan^{-1}\left(\frac{pos_{traj_y} - pos_{current_y}}{pos_{traj_x} - pos_{current_x}}\right) \end{aligned} \tag{5.10}$$

5.3.4 Local Goal Distance Cost

Similar to the heading costs, a local goal distance cost measures the translational difference between the local goal and the end position of the trajectory candidate.

$$\begin{aligned} c_{d_{local}} &= d_{local-traj_i} \\ d_{local-traj_i} &= \sqrt{(pos_{local_y} - pos_{traj_y})^2 + (pos_{local_x} - pos_{traj_x})^2} \end{aligned} \tag{5.11}$$

5.3.5 Global Goal Distance Cost

A global goal distance cost measures how closely the end pose of the trajectory candidate will be to the global goal, which balances with the local goal distance cost so that the planner also favors the trajectory that tries to reach the destination.

$$\begin{aligned} c_{d_{global}} &= d_{global-traj_i} \\ d_{global-traj_i} &= \sqrt{(pos_{global_y} - pos_{traj_y})^2 + (pos_{global_x} - pos_{traj_x})^2} \end{aligned} \tag{5.12}$$

5.3.6 Obstacle Cost

An obstacle cost measures the potential for collision of the poses in the trajectory candidate. In addition to the local reactive controller in §4.5, additional safety constraints should be added such as a minimum distance that the trajectory can travel.

First, our trajectory collision checking approach operates in the perception space. Since the trajectory generation method in the proposed global navigation framework that constructs candidates based on the state of the robot does not guarantee that all of the 3D Cartesian poses in the trajectory candidate will be projected in the perception space. Hence, the obstacle cost returns -1 if the planner is unable to evaluate the trajectory in the perception space.

Our first criteria of validating the trajectory is that at least one pose in the trajectory must be projected in the perception space for evaluation. For the area that is in front of the robot but outside of the field of view, the planner assumes that poses in this area are safe. Finally, the planner utilizes the safe travel distance as a principal evaluation metric in the obstacle cost function. Since our objective function is to minimize the sum of cost functions explained above, we compute the obstacle cost by taking an inverse of the safe travel distance.

$$c_{obstacle} = \textit{ObstacleCost}(T_i) \tag{5.13}$$

where $\textit{ObstacleCost}(traj)$ is defined in Algorithm 3.

Algorithm 3 Obstacle Cost evaluation

procedure OBSTACLE COST($Trajectory$)
  $U \leftarrow$ *Image Width*
  $V \leftarrow$ *Image Height*
  $d_{safe} \leftarrow 0.01$
  $x_{current} \leftarrow x_0$
  $y_{current} \leftarrow y_0$
  $proj_{once} \leftarrow false$
  for each $(x_i, y_i, z_i) \in Trajectory$ **do**
    $u, v \leftarrow imageSpaceProjection(x_i, y_i, z_i)$ → §4.2
    if $(0 <= u < U) \wedge (0 <= v < V)$ **then** → Point in the perception space boundary
      if $collision(u, v)$ **then** → §4.4
        if $proj_{once}$ **then**
          return $\frac{1}{d_{safe}}$
        else
          $Return - 1$
      else
        $d_{safe} \leftarrow d_{safe} + \sqrt{(x_i - x_{current})^2 + (y_i - y_{current})^2}$
        $x_{current} \leftarrow x_i$
        $y_{current} \leftarrow y_i$
    else → Point not in the perception space boundary
      if $(0 <= u < U) \wedge (v >= V)$ **then** → Out of the perception space boundary, but in front of the robot
        $proj_{once} \leftarrow true$
        $d_{safe} \leftarrow d_{safe} + \sqrt{(x_i - x_{current})^2 + (y_i - y_{current})^2}$
        $x_{current} \leftarrow x_i$
        $y_{current} \leftarrow y_i$
      else → Out of the perception space boundary and the area is unknown
        if $proj_{once}$ **then**
          return $\frac{1}{d_{safe}}$
        else
          $Return - 1$
  return $\frac{1}{d_{safe}}$

5.3.7 Trajectory Selection

The robot follows a trajectory that minimizes the overall cost function until the next control loop.

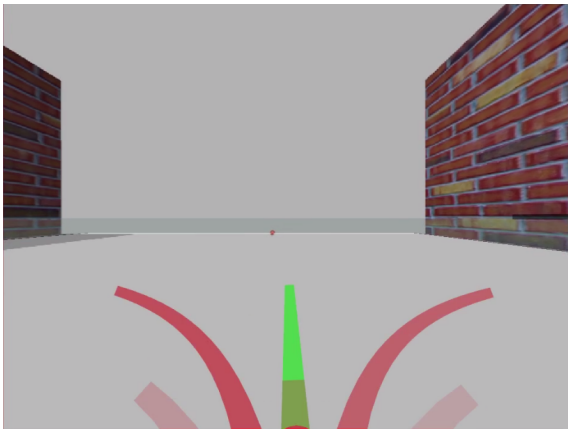
$$c(T_i) = c_{oscillation} + \alpha c_{\theta_{local}} + \beta c_{\theta_{global}} + \gamma c_{d_{local}} + \lambda c_{d_{global}} + \mu c_{obstacle} \tag{5.14}$$

where $\alpha, \beta, \gamma, \lambda,$ and μ are scale factors for each cost function. Again, the planner does not select a trajectory if any of cost function returns a negative value. Once the best trajectory T^* is found, the planner provides the corresponding velocity command (v_x, v_y, v_θ) to the robot drive controller.

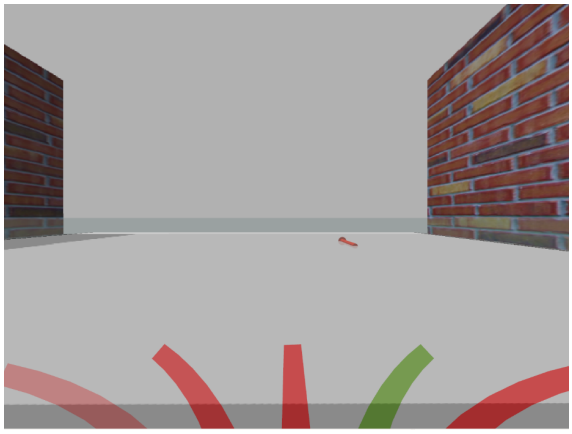
We show how the overall cost affects local path planning in 5.3. In Figure 5.3a, we visualize the trajectory candidate set and the selected candidate in the perception space. For visualization purposes, only two samples from from both v_x and v_θ ($v_x \in (0.4, 0.25m/s)$ and $v_\theta \in (0, 0.25\ radian/s)$ are used to generate the trajectory candidate set. Trajectories with positive values are colored red, weighted by the overall cost $c(T_i)$. The smaller the cost, the intensity of the projected trajectory gets larger. The selected trajectory is colored green.

In the figure, we provide a global goal location in front of the robot. As a result, we observe that the planner selects the trajectory that drives the robot forward with the faster speed, $v(0.4m/s, 0\ radian/s)$. Another note is that the planner evenly favors trajectories with the same v_θ for each v_x because the cost difference between these two candidates are very close since all the cost functions return similar values. In contrast, the trajectories that drive the robot slower are least favored due to smaller values returned from local and global goal distance costs.

In Figure 5.3b, we sample 5 trajectories using a same linear velocity but different v_θ. In this scenario, we provide a global goal location to the right side of the robot nose marked as a red arrow. As a result, we observe that the planner selects the trajectory that aligns its heading to the goal with the closest distance. While the trajectory in the middle gets the second lowest value from the overall cost function, another note is that the planner roughly favors the trajectory in the second left and far right. One interpretation is that the trajectory in the second left gets favored from the heading cost functions since the heading of the robot at the end of the trajectory will head towards the destination. In contrast, the trajectory in the far right gets favored because the translational difference will be smaller.



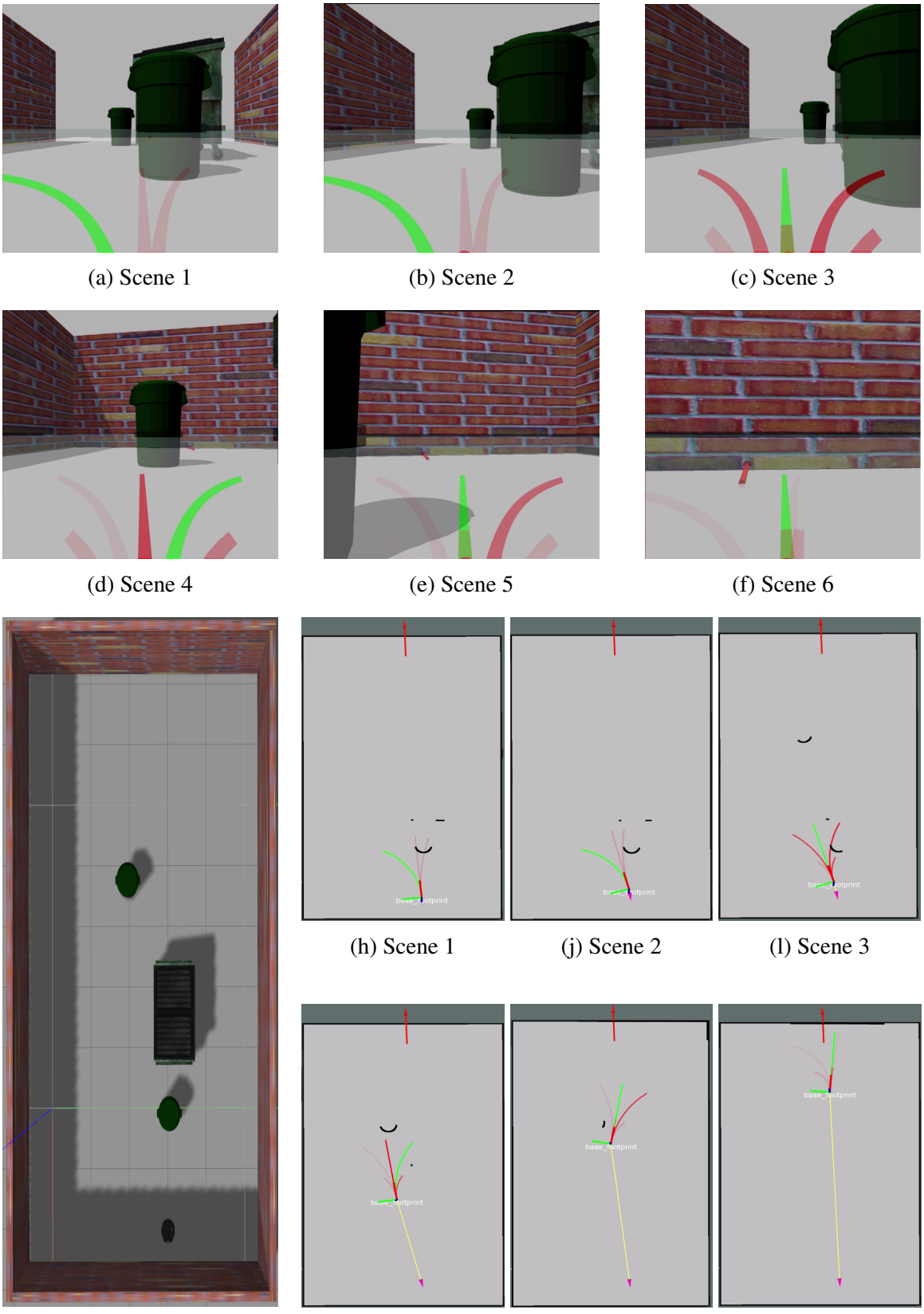
(a) Projection of candidate trajectories in the perception space weighted by the overall cost, Goal Front



(b) Projection of candidate trajectories in the perception space weighted by the overall cost, Goal Right

Figure 5.3: Trajectory projection in the perception space

The next simulation experiment contains two trash bins and one dumpster as shown in Figure 5.4g. A goal of the navigation task is shown as a red arrow in Figure 5.4h. Trajectories with the positive overall cost are colored red weighted by the overall cost as in 5.3 and projected in both the world and image representation. The selected trajectory is colored green. In Scene 1, the planner selects a trajectory that turns left because the two other trajectory candidates collide with the obstacle. The planner selects the same trajectory in Scene 2 even though the trajectory in the middle does not appear to collide with the obstacle. This is due to the robot model projection based collision checking described in §4.4. By projecting each pose in the trajectory, the planner detects that the right side of the robot model collides with the obstacle. In Scene 3, the trajectory no longer collides with the obstacle and is selected. In Scene 4 and 5, the planner then tries to return to the optimal path (*i.e.*, global plan) because no obstacle is found in the perception space. Finally in Scene 6, the planner selects the straight trajectory which will take the robot within the tolerance distance of the goal, thereby completes the navigation task.



(a) Scene 1 (b) Scene 2 (c) Scene 3

(d) Scene 4 (e) Scene 5 (f) Scene 6

(g) World B (h) Scene 1 (j) Scene 2 (l) Scene 3

(i) Scene 4 (k) Scene 5 (m) Scene 6

Figure 5.4: Navigation Task with obstacles

CHAPTER 6

EXPERIMENTAL RESULTS

6.1 Stixel Construction

Parameters that we use to build the Stixel representation are the following:

Table 6.1: Stixel world parameters

Parameter	Value
Expected maximum object height	100cm
Maximums disparity d_{max}	128 pixels
Neighbouring Pixel Size Δ_{max}	10 pixels
Stixel Width	1 pixel

6.2 Statistical Data

In Table 6.2, we compare the processing time of each stage in Stixel construction using our approach to that of the full depth map computation based approach used in [32]. For depth map computation, we use OpenCV 2.4 Semi-Global Block Matching using SAD over 9x9 pixels with winner−take−all strategy. Based on the estimated depth map, we assign a deterministic disparity value at each pixel and compute the $u\text{-}disparity$ and $v\text{-}disparity$ representations.

As already mentioned in §3.1 and §3.2, the proposed Stixel estimator reduces the computation time by limiting the search window of the Stixels based on the cost volume matrix and ground plane estimation. From the table, we show that our Stixel estimator can construct the Stixel representation at a rate of 29Hz for our camera's maximum image resolution of 1080x720 pixels. Skipping the stage of computing the full depth map and only comparing the disparity cost matrix of the region of interest for the Stixel estimation, we see that the proposed approach is significantly faster than the full depth map based approach.

Table 6.2: Computation time of Stixel world

Dimension	Stage	Cost Volume Approach	Depth Map Computation
		Time(ms)	Time(ms)
320 x 240	Ground Plane Estimation	0.54	4.98
	Stixel Estimation	4.16	24.54
640 x 480	Ground Plane Estimation	2.87	10.27
	Stixel Estimation	12.65	97.48
1080 x 720	Ground Plane Estimation	8.87	68.19
	Stixel Estimatioin	25.76	470.02

In Table 6.3, we show the processing time for evaluating a trajectory candidate that consists of 80 sampled poses using our Stixel-based collision checking approach compared to the depth image based collision checking algorithm, PiPS [28]. From the table, the robot model projection time using the Stixel-based approach is significantly faster than PiPS. While PiPS includes pixelwise depth value comparisons of the projected robot model for the collision checking, our approach compares one $d_s^*(u)$ for each column along the horizontal boundary of the robot model, allowing it to evaluate each pose much faster. As a size of the robot model increases, the computation time using PiPS will increase quadratically while our approach increases linearly.

Table 6.3: Computation time of model projection based approach

Approach	Approach	Time(μs)
Stixel	Rectangular Model Projection	832
	Cylindrical Model Projection	2845
Smith and Vela	Rectangular Model Projection	1764
	Cylindrical Model Projection	8724

6.3 Simulation Experiments

While real-world experiments are relatively challenging to reproduce same environments, simulation serves to create highly repeatable and consistent test conditions for evaluating a variety of planning implementations. Therefore, we evaluate our Stixel-based navigation framework quantitatively by comparing with traditional navigation systems.

6.3.1 Simulation System

We use Gazebo version 7.8.1 to simulate the environment and mobile vehicle and Robot Operating System (ROS) version Indigo to interface with the Gazebo simulation. Simulations run on a Intel i7-4770 3.4GHz 8-core processor with 16GB RAM. The simulated mobile robot is a Turtlebot 2 with a stereo camera mounted on the second plate of the robot. The stereo camera is located at $(0.05, 0, 0.21)$ meters in the robot base frame. We show the mobile robot that we use for experiments in simulation in Figure 6.1. Note that the robot also equips the depth camera for use by other types of navigation planners.

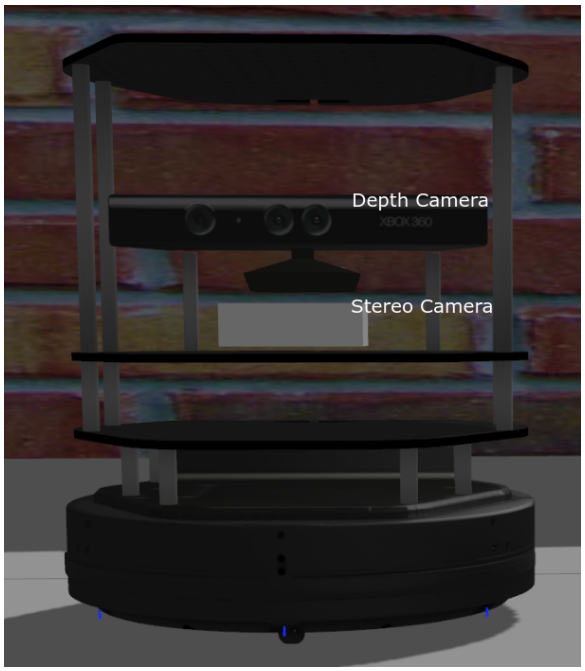


Figure 6.1: Mobile vehicle in simulation

In Table 6.4, we list important configuration parameters of the simulated stereo camera used during the experiments.

Table 6.4: Camera parameters in simulation

Parameter	Value
Camera Driver	ROS Multisense sl
Baseline	12cm
Image Dimension	640x480 RGB
Image Rate	30 FPS
Focal Length	554
Horizontal FOV	120°

To effectively compare each controller under same configurations and limitations, we utilize common parameters for all the controllers. We show the parameters in Table 6.5.

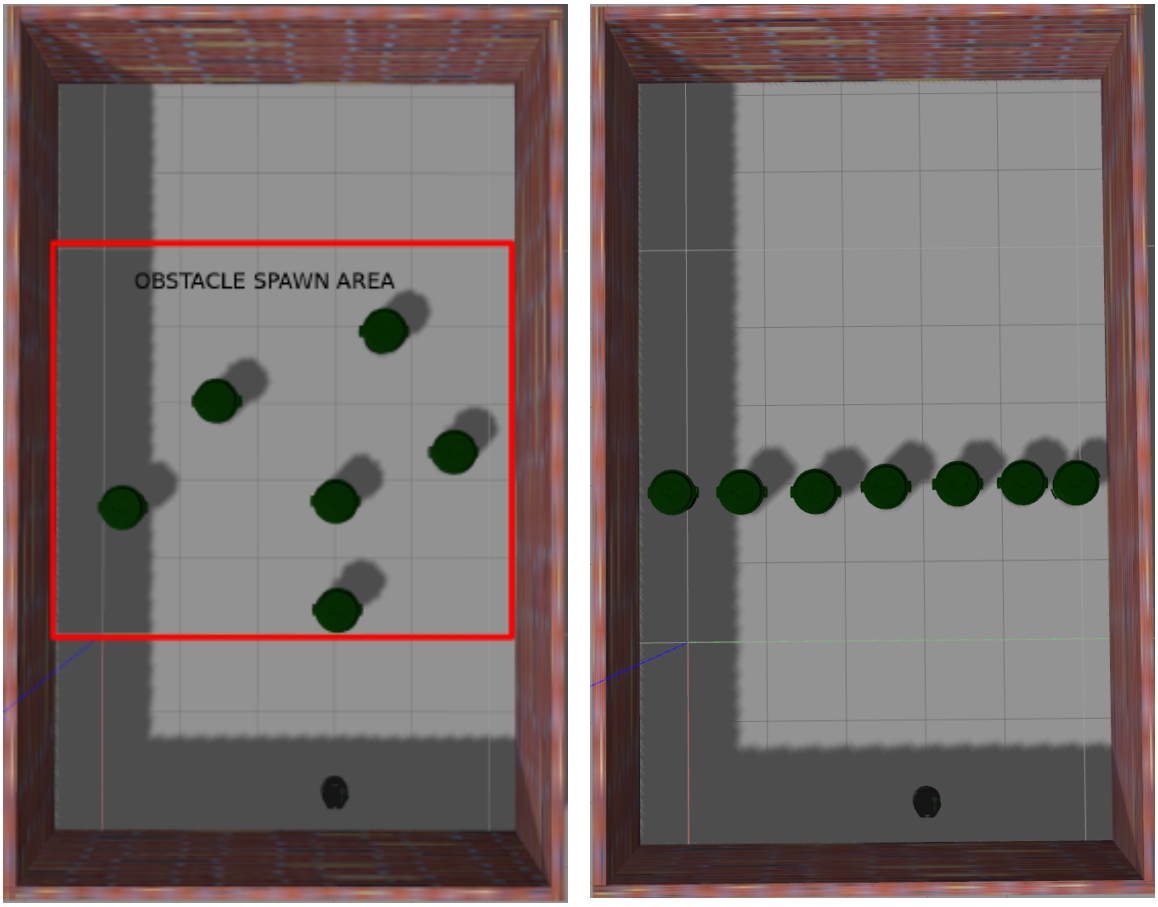
Table 6.5: Controller parameters in simulation

Parameter	Value
Acceleration Limit $(a_{lim_x}, a_{lim_y}, a_{lim_\theta})$	(2.5m/s^2, 2.5m/s^2, 3.2 radians/s^2)
Velocity Limit $(V_{max_x}, V_{max_y}, V_{max_\theta})$	(0.5m/s, 0.5m/s, 0.5 radians/s)
Trajectory Simulation Time t_s	5 seconds
Velocity Samples	200
Oscillation cost parameter $(d_{trans}, \theta_{rot})$	(0.05, 0.2)
Objective function parameters $(\alpha, \beta, \gamma, \lambda, \mu)$	(24, 32, 24, 32, 50)

6.3.2 Rectangular World

In this experiment, we explore the robustness of the reactive obstacle avoidance capability of the Stixel navigation framework. The objective of the navigation task is to move from one end of the world to the other end without crashing. In a 10 x 6m rectangular world, the robot has to travel approximately 8 meters forward. We repeatedly execute navigation tasks in randomly generated scenes. We iterate the local scene for each controller by giving the same destination. For each local scene, there are a number of barrels that are randomly placed in the zone shown as the red rectangle in Figure 6.2a. As more numbers of barrels are placed in the fixed size of the rectangle, we expect that the planner which performs

more robust and accurate obstacle avoidance will score higher success rate of the navigation tasks. Also, there is no guarantee that there exists any traversable path that the planner can choose because locations of barrels are randomly chosen (*i.e.*, if all barrels are lined up each other horizontally as shown in Figure 6.2b). We set the number of barrels to 3, 5, and 7. For each quantity of barrels, 100 scenes are created.



(a) Rectangular world scenario example

(b) Local scene that is guaranteed to fail

Figure 6.2: World level view of Rectangular World

For quantitative evaluation, We compare our Stixel-based navigation framework with four different planners. Three of them operate on the 2D Cartesian grid with a laser scan data as their primary source. One is a vision-based planner that uses the depth image. We populate the laser scan data by taking a depth image from the Kinect camera. The planners that we use in this experiment are the following:

- Dynamic Window Approach Local Planner (2D Cartesian, Laser)[20]
- Elastic Band Local Planner (2D Cartesian, Laser)[21]
- Timed Elastic Band Local Planner (2D Cartesian, Laser)[22]
- PiPS Local Planner (Depth Space, Depth Camera)[28]
- Stixel Local Planner (Stixel Space, Stereo Camera)

We show the success rate of navigation tasks in Figure 6.4a. From the figure, we observe that the Stixel-based planner is capable of navigating through densely placed obstacles compared to traditional navigation planners. The TEB planner scores the highest success rate and one interpretation is that the planner constructs more safe and robust exit plan in the dead zone. When the robot gets stuck in the dead zone, the area which is impossible to navigate (*i.e.*, Figure 6.3), the TEB planner constructs a path that drives the robot backwards following the previous paths that the robot took. After the robot moves backwards for certain duration, the planner constructs a new global plan with the knowledge of the dead zone.

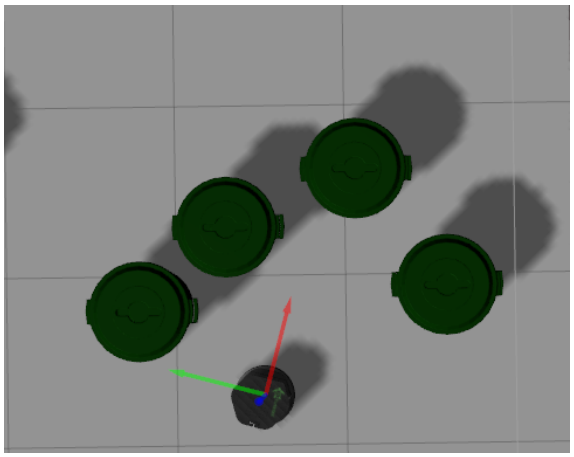


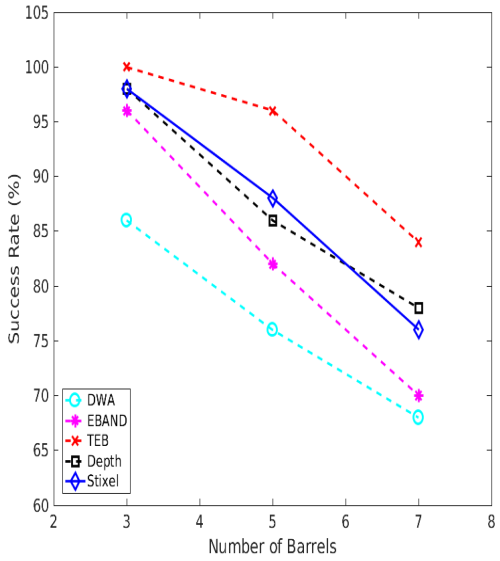
Figure 6.3: Dead zone example

For the Stixel planner, however, this type of approach remains as a future work since the underlying assumption of our perception space path planning is that the planner does

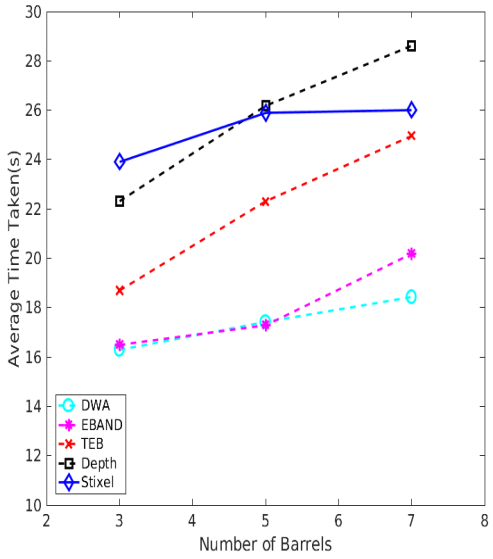
not convert and update the obstacle observation in the world representation. In our current implementation, the exit strategy when the Stixel planner cannot find a valid path is to rotate 180 degrees and reconstruct a new global plan so that the robot can explore the local scene much longer until it finds the traversable path. The other 2D Cartesian local planners also implement a similar type for their exit strategy, rotating 720 degrees to update the local costmap and reconstruct the new global plan.

The figure also shows that the success rate of the Stixel planner is the second highest scored planner despite the fact that 2D Cartesian planners still accumulate the local observations in the world representation. This is because their update approaches are pointwise based expansion and these approaches significantly simplify the 3D geometry of the robot model, which over-expands the obstacles. In the cluttered environment like the Rectangular World, over-expansion of the obstacles can cause the navigation system to fail to find a path that navigates through a narrow space between obstacles.

In Figure 6.4b, we show the computation time of the Stixel navigation by comparing the average time taken for each planner when the navigation task is successful. While the average time of the Stixel planner seems much longer than other 2D Cartesian planners, Cartesian planners update the world representation based on the populated laser scan data, which is a simple 1D line search from the depth image. In contrast, the Stixel navigation planner includes various subprocesses such as Stixel world construction and robot model projection-based collision checking. Even with many subprocesses, the average completion time of the Stixel planner is comparable to the TEB planner which updates the world representation significantly faster and is the only planner that scores the higher success rate than the Stixel planner on average.



(a) Controller comparison (Success rate)



(b) Controller comparison (Average time to complete)

Figure 6.4: Rectangular World Experimental Result

6.3.3 Random World

While navigation tests in the Rectangular World evaluate robust obstacle avoidance capability in densely located obstacles, the Random World evaluates how well each local path planning approach is integrated with the global navigation framework. Unlike the Rectangular World, the Random World contains different types of objects which can identify the effectiveness of model projection-based collision checking and the quality of our Stixel construction approach. This 20 x 20m world is evenly divided into nine different sectors where the objective of each task is to travel between the selected start and goal sectors, denoted as red dots in Figure 6.5a. Start and goal sectors are randomly selected for each test. Based on the experimental results from the Rectangular World, we choose three best planners. We run 100 navigation tasks for each controller.

- Timed Elastic Band Local Planner (2D Cartesian, Laser)
- Depth PIPS Local Planner (Depth Space, Depth Camera)

- Stixel Local Planner (Stixel Space, Stereo Camera)

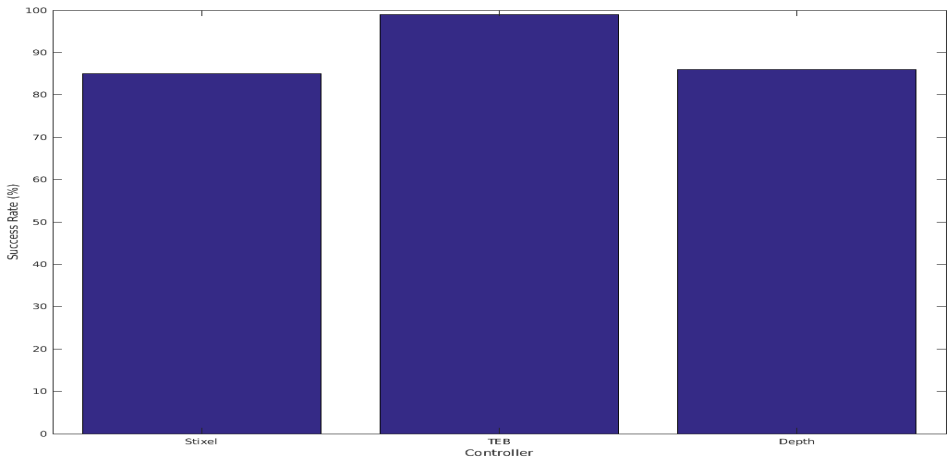
We use the same robot configuration parameters as in the Rectangular World experiments.

(a) Random World Top View (b) Random World Side View

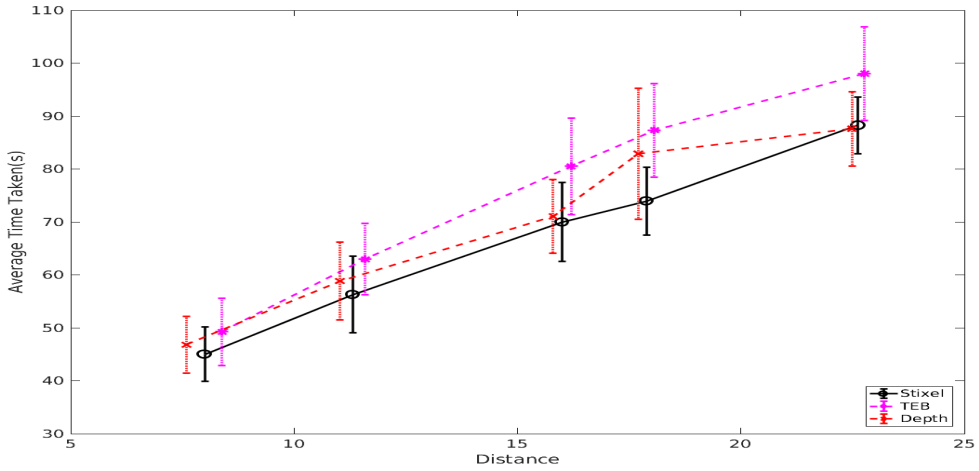
Figure 6.5: World View of Random World

In Figure 6.6a, we show the success rate of the three navigation planners. The Stixel navigation planner achieves a similar success rate to the depth based navigation planner. Even though our approach does not maintain local obstacle information, its success rate is comparable to that of the TEB planner. We show a disadvantage of accumulating local obstacle observations in the world representation in Figure 6.6b. Unlike the Rectangular World, each navigation task in Random World has a different travel distance for each test. The average travel time of the TEB planner increase more rapidly with the travel distance of the navigation task than that of the vision-based planners. One interpretation is because the planner has to accumulate the observations in the world representation as the robot travels, which increases the amount of information the planner has to maintain. If this type of 2D Cartesian planners uses their primary source from the vision data instead of the laser scan, the planner has to include the additional process of converting the vision data into the Cartesian grid-compatible data type such as a point cloud. Hence, the update process will take significantly longer and the planner will have to sacrifice it by decreasing the local

planning frequency, which eventually causes either longer travel time or unsafe navigation.



(a) Controller Success Rate



(b) Average Time Taken per distance

Figure 6.6: Random World Experimental Result

6.4 Real World Implementation

In this section, we implement our navigation framework on a real world system to evaluate the navigation performance qualitatively.

6.4.1 Real World System

In real world-based experiments, we again use the Turtlebot 2 as our base mobile platform and ROS Indigo for the hardware and software interface. We run the navigation tasks on an Intel i7-6500U 2.5GHz 4-core processor with 8GB RAM for processing hardware. A stereo camera that we use in the experiments is a DUO M stereo camera mounted under the third plate as shown in Figure 6.7. The translation is $(0.076, 0, 0.29)$ in the robot base frame.

Figure 6.7: Real world mobile vehicle

Important parameters of stereo camera are listed in Table 6.6.

The controller configuration parameters that we use for the real time testing are listed in Table 6.7.

Table 6.6: Real world stereo caemra parameters

Parameter	Value
Camera Driver	Custom Implementation
Baseline	3cm
Image Dimension	752x480 Gray
Image Rate	30 FPS
Focal Length	300
Height	30cm from the ground
Pitch	0°
Horizontal FOV	170°

Table 6.7: Real world controller parameters

Parameter	Value
Acceleration Limit $(a_{lim_x}, a_{lim_y}, a_{lim_\theta})$	(2.5m/s^2, 2.5m/s^2, 3.2 radians/s^2)
Velocity Limit $(V_{max_x}, V_{max_y}, V_{max_\theta})$	(0.45m/s, 0.45m/s, 0.45 radians/s)
Trajectory Simulation Time t_s	5 seconds
Velocity Samples	25
Oscillation cost parameters $(d_{trans}, \theta_{rot})$	(0.05, 0.2)
Objective function parameters $(\alpha, \beta, \gamma, \lambda, \mu)$	(24, 32, 24, 32, 75)

6.4.2 Stixel and Path Evaluation

We show example trajectory evaluations in the Stixel representation in Figure 6.8. Due to the short baseline of our stereo camera, the effective depth range is only from 0.3 to 2 meters. For visualization purposes, we only visualize five trajectories constructed using the departure angle using [28] so that the planner can project all the trajectories in the perception space to explore the performance of the proposed Stixel-based collision checking.

First, as seen in Figure 6.8, the Stixel estimator detects most vertical objects that stand from the ground plane within the effective depth range. Also, trajectory evaluation using the robot model projection approach performs well in the real world.

Throughout the experiment, we find several failure cases of Stixel estimation. There failure cases mostly happen on the objects that have a lack of textures. From Figure 6.9, the Stixel estimator is not accurately constructing the Stixels over the region that does not have enough features such as a wall. Our cost volume matrix based Stixel estimation assumes

that each object has unique textures and intensities that differentiate it from other types of objects. If there is not enough uniqueness in the object, the Stixel estimator is not able to construct the accurate Stixel representation. Since the stereo matching mostly focuses on outdoor scenes where each object has unique texture, this weakness is somewhat less important than other navigation-related limitations.

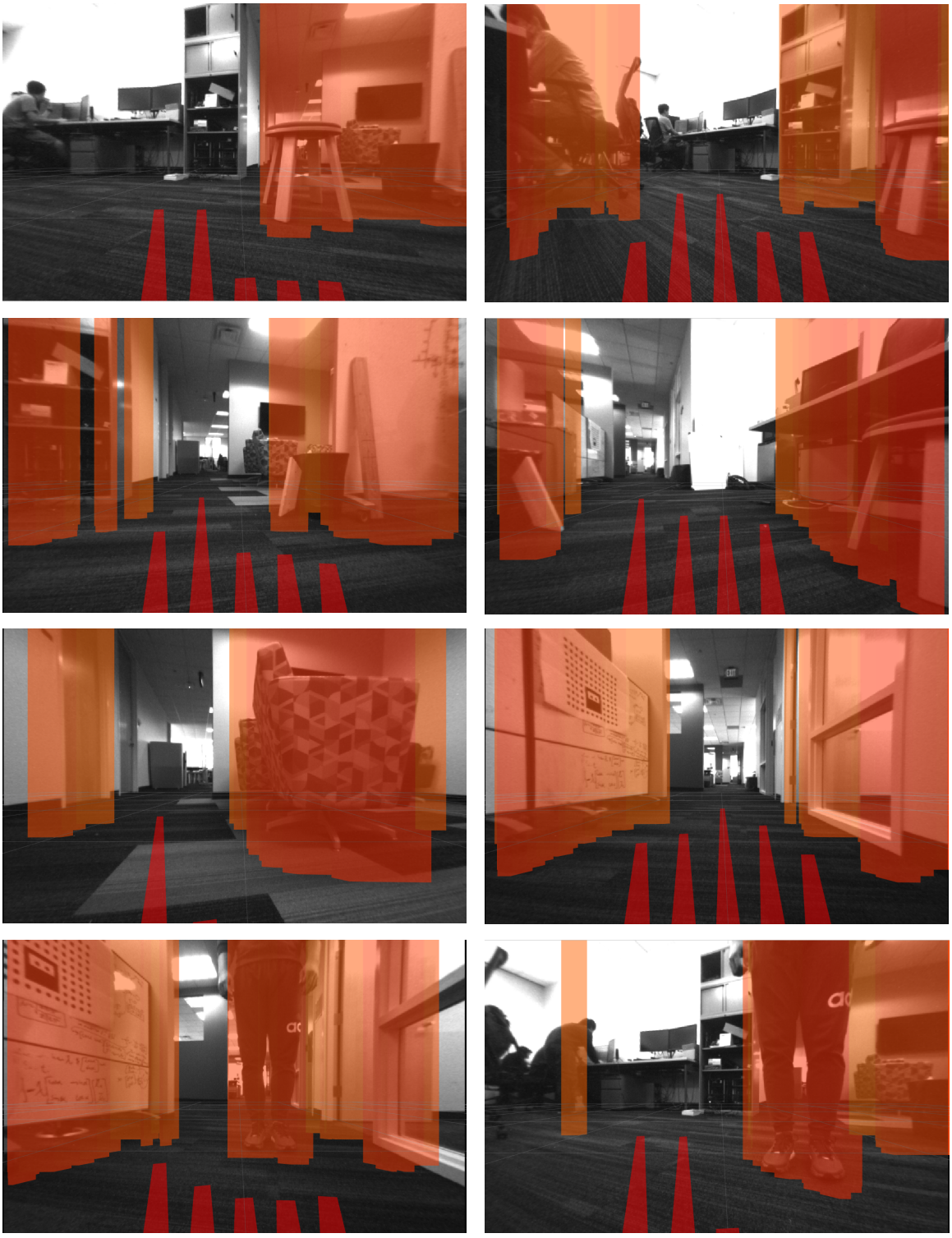


Figure 6.8: Trajectory Evaluation in the Stixel representation

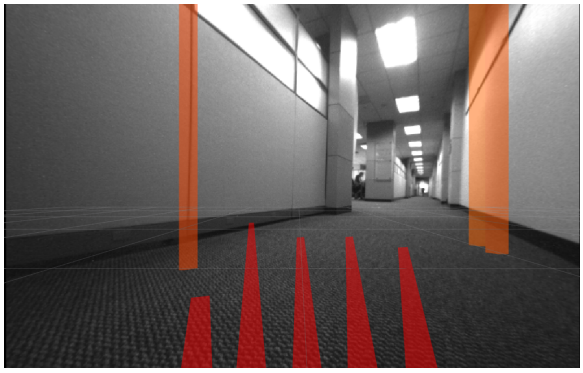


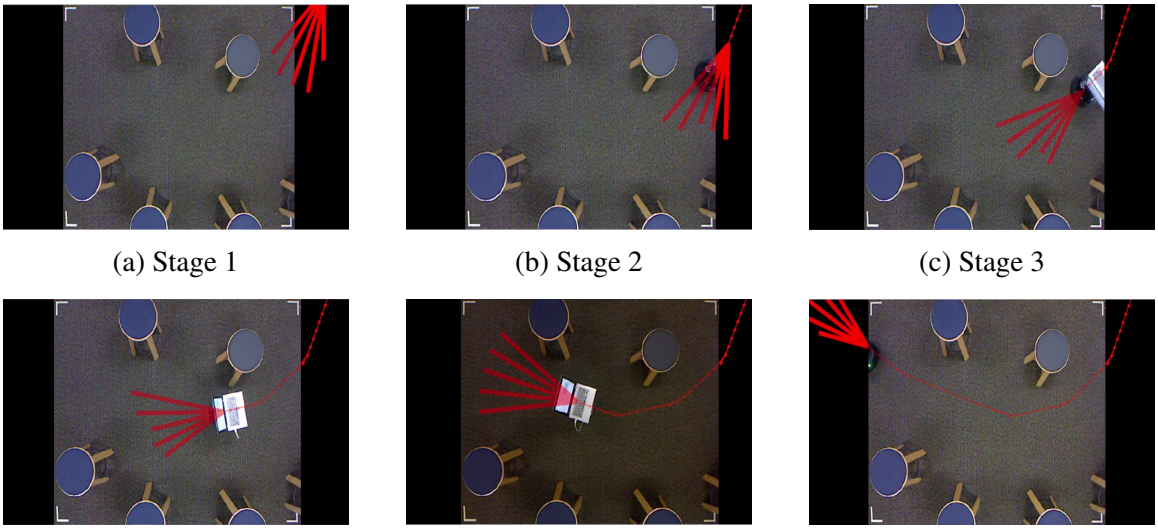
Figure 6.9: Failure case of Stixel estimation

6.4.3 Navigation Task

Navigation with overhead view

In this experiment, we explore the capability of the reactive obstacle avoidance in real world navigation using an overhead view. We restrict the navigation area to be a 3.2x2.4 meter of rectangular region within the filed of view of an overhead camera in order to capture the entire navigation task.

We show a subset of overhead views of the navigation task for one scenario in Figure 6.10. The Turtlebot starts off the screen at the upper-right (just past the barely visible white marker) and is directed to travel to another off-screen location at the upper-left of the image. We visualize each trajectory candidate based on its safe travel distance from the base of the robot. Since the destination of the navigation is upper-left of the image, the robot tries to stay to the upper side as long as there are no obstacles. More detailed overhead image sequences and the corresponding Stixel constructions and path evaluations can be found in Appendix A.



(a) Stage 1 (b) Stage 2 (c) Stage 3

(d) Stage 4 (e) Stage 5 (f) stage 6

Figure 6.10: Real world navigation task in a rectangular region

We test our approach 20 times with different obstacle positions. For 14 cases, the navigation task from the start to goal is successful. In 2 of 6 failure cases, the valid path is not found during the navigation because of incorrect Stixel construction. For the remaining 4 cases, the Turtlebot collides with obstacles located just outside of the stereo camera's field of view.

Global Navigation

In this experiment, we repeatedly test the Stixel-based global navigation framework to explore the floor. From the primary experimental results from §6.4.2, we discover that the cost volume matrix approach does not perform well over textureless regions such as a wall. Therefore, we provide these regions to the navigation planner as the prior knowledge enabling it to construct global plans that avoid those walls. The map used in the experiment is shown in Figure 6.11.

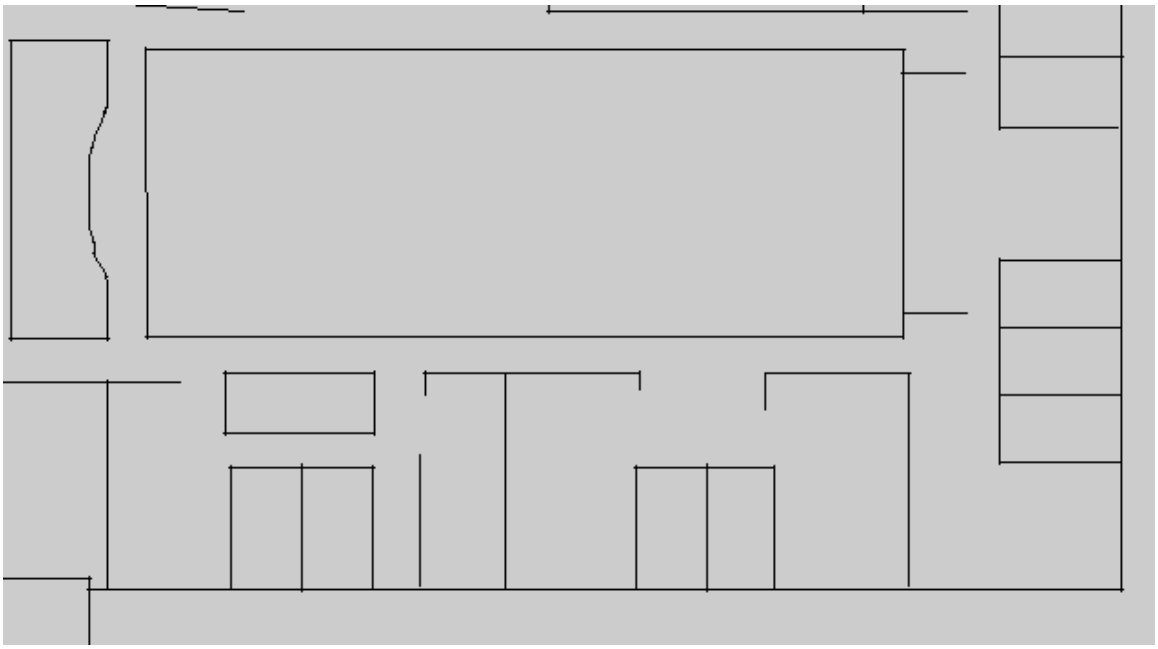
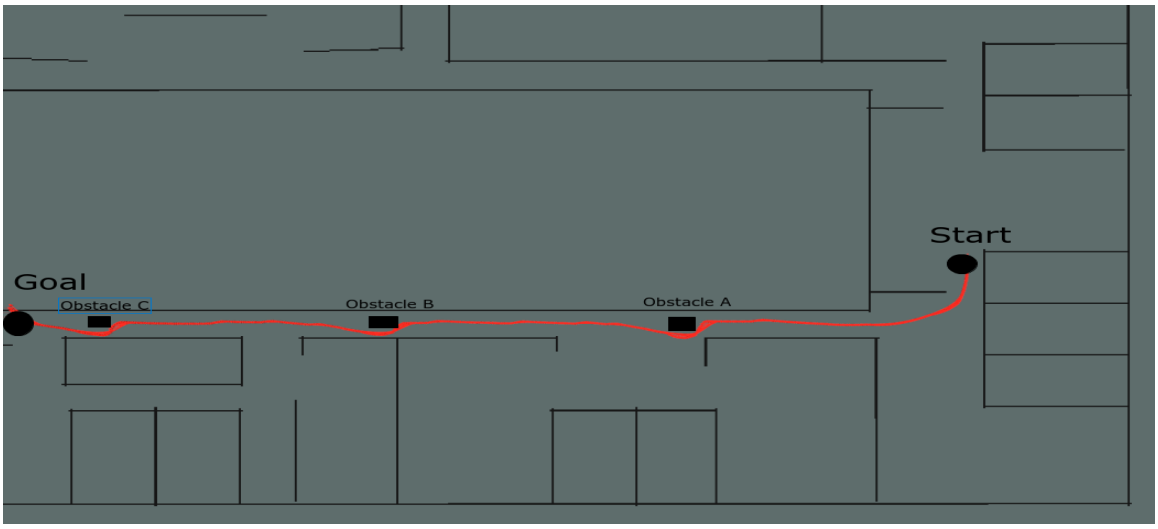
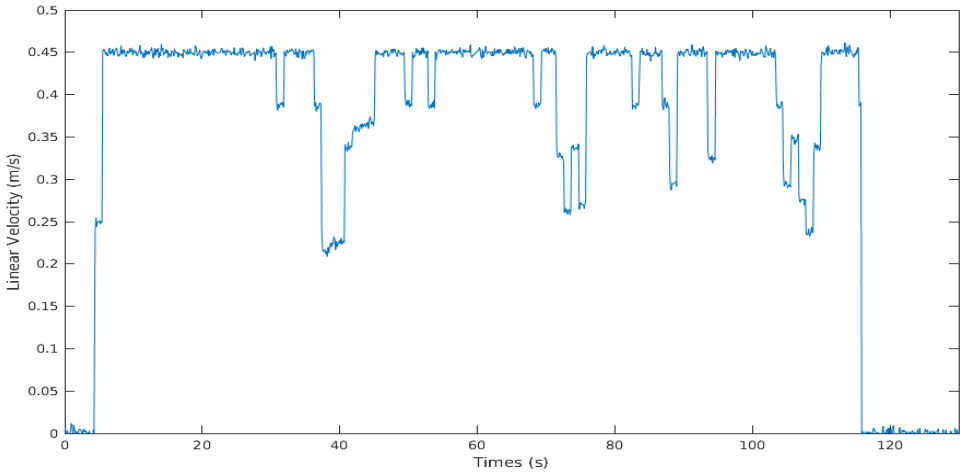


Figure 6.11: Real World Map

We capture the 2D world representation view of one navigation task in Figure 6.12 For this task, the objective of the navigation is to traverse through a narrow corridor where there are three unknown obstacles along the way. For the task, the Stixel planner only knows the position of its current location and the destination. We illustrate the start and goal locations and the positions of obstacles in Figure 6.12a as well as the odometry history of the robot recorded during the navigation task. From the odometry history, we see that the Stixel planner avoids the obstacle safely while trying to reach to the destination as fast as possible. In Figure 6.12b, we show the corresponding linear velocity changes during the navigation. Decreased linear velocities are the result of increased angular velocities from avoiding obstacles. As soon as the planner safely avoids the obstacles, the planner then tries to reach its maximum velocity again.



(a) Odometry layered in the 2D Cartesian Grid with start and goal location



(b) Linear velocity over time

Figure 6.12: Real world navigation in a corridor

In this experiment, we test our Stixel-based global navigation planner framework in different locations 10 times. For 6 cases, the navigation tasks from start to goal are successful. 3 of 4 failure cases are due to a collision caused by the inaccurate Stixel construction. One last failure case occurs due to an object being outside of the field of view when the planner turns a corner.

6.5 Limitations

In this section, we discuss several limitations discovered during experiments.

6.5.1 No Obstacle History

For scenarios with sparsely located obstacles, local path planning without maintaining obstacle information performs relatively well and is robust compared to traditional vision-based navigation planners that include the conversion process. For scenarios that require global replanning, such as a dead zone exit, the navigation planner needs to know the local obstacle information in order to construct the new global plan. The traditional navigation planners can reconstruct the global plan using the information in the world representation that is updated at every control stage. In contrast, our planner only has knowledge of obstacles that are in the image space at the current control stage. Therefore, the exit strategy that we implement is to make the robot rotate and let it explore the local scene more extensively. We observe that the TEB planner can drive the robot backwards safely based on the recent obstacle information, which can help the robot to exit the dead zone more efficiently. In order to implement similar feature as the TEB planner, we can implement an approach to extend the visual obstacle information locally in the perception space (*i.e.*, Local visual memory [48]).

6.5.2 Limited Field of View

Throughout the experiments, we find some failure cases of the Stixel navigation planner colliding with obstacles when the robot tries to make a sharp turn around an obstacle close to the robot. It is one of canonical problems in vision-based navigation where obstacles are not seen in the image plane and therefore the navigation planner cannot reactively avoid them. In Figure 6.13, an obstacle is right next to the mobile vehicle but is outside of the camera's field of view. Assuming the goal location is placed on the far right side with

respect to the robot, the navigation system chooses to go to the right because no obstacle is found in the perception space. Since it may never come into field of view during the navigation, a collision will likely occur. We can partially solve this problem by fusing the vision based collision checking with other types of sensor data.

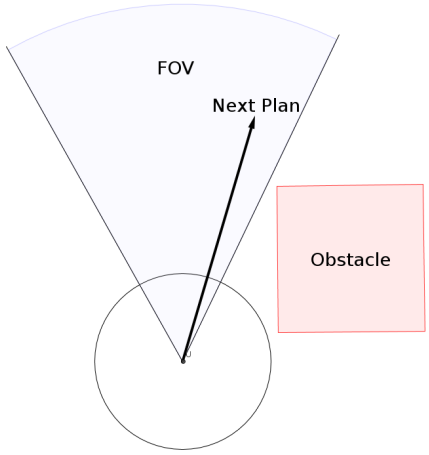


Figure 6.13: Limitaion of Vision Based Navigation

6.5.3 Stereo Matching Over Textureless Area

Stereo matching algorithms usually assume that objects will have unique textures. As we discovered, from the inaccurate Stixel World constructions over the wall (shown in Figure 6.9), regions that have large stereo ambiguity also affect the performance of the navigation. We can address this type of limitations by implementing more sophisticated stereo matching algorithms such as [49] with a trade off of higher computation time, which is also critical for the autonomous navigation.

CHAPTER 7
CONCLUSION & FUTURE WORK

7.1 Conclusion

We have presented an approach and a system framework that allows robust and non-myopic vision-based navigation system that performs path planning in the medium level representation called Stixel World. Our system does not interpret and convert any information from the perception space into a world representation to improve the robustness in local path planning. The key factor that enables our system to perform local path planning in the perception space without conversion is projecting the 3D physical geometry of the mobile robot into the Stixel representation by comparing the estimated Stixel disparity and the depth of the projected model.

In order to evaluate the Stixel navigation planner quantitatively and qualitatively, we tested our approach both in simulation and the real world. As a result of our improved computation time in Stixel world construction, the navigation system in both environments was able to maintain approximately 30 frames of the Stixel world per second even at our camera's maximum resolution of 1080x720 pixels. From experimental results from both simulation and the real world, we showed that the Stixel navigation planner is suitable for the autonomous navigation of the mobile robot.

7.2 Future Work

The current Stixel representation lacks an accurate Stixel boundary detection, $d_s^*(u)$ if the object does not have unique texture and color, which makes the navigation system vulnerable to collision for those objects. We can address this problem by implementing more sophisticated stereo matching algorithms with the trade-off of a decreased planning fre-

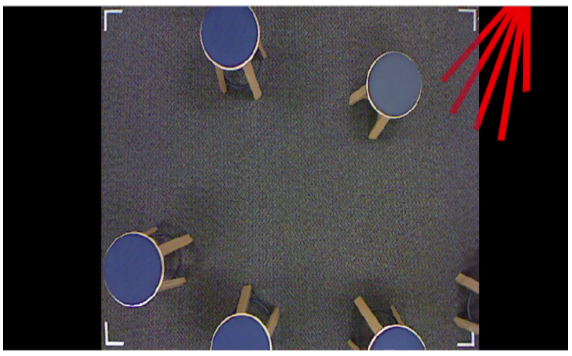
quency. Another resolution is to integrate the stereo camera with other types of visual data such as a Time-of-Flight (ToF) camera and depth camera. ToF camera has an advantage in depth estimation over textureless objects while the stereo camera performs unsatisfactorily. Depth camera can give precise depth information indoor.

Also, the Stixel navigation planner can be implemented on a large-sized real road vehicle since the Stixel World is first proposed for detecting pedestrians and obstacles on the real road. By recognizing pedestrians that are expected to be dynamically moving in the local scene, the navigation planner is able to distinguish between dynamic objects that have higher priority to be avoided and small static objects that might not be significantly hazardous.

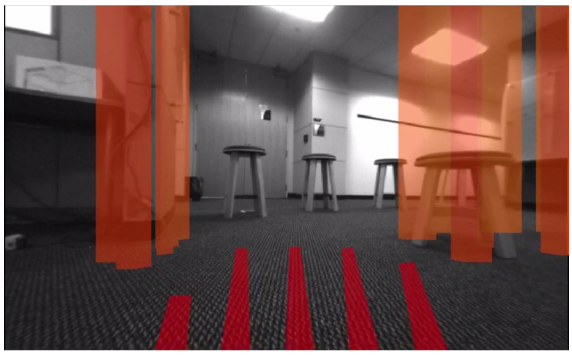
Appendices

APPENDIX A

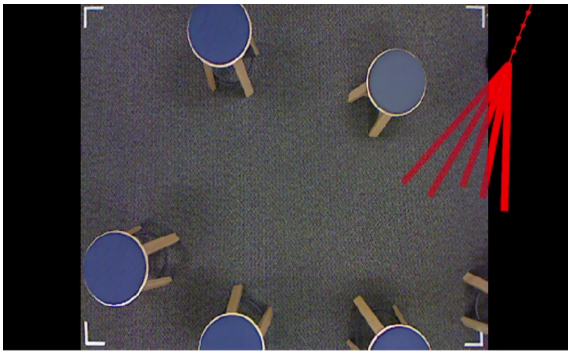
DETAILED SCREENSHOTS OF REAL TIME NAVIGATION TASK



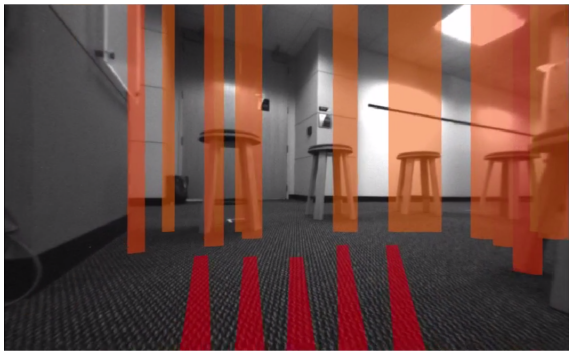
(a) Overhead view 1-Navigation started



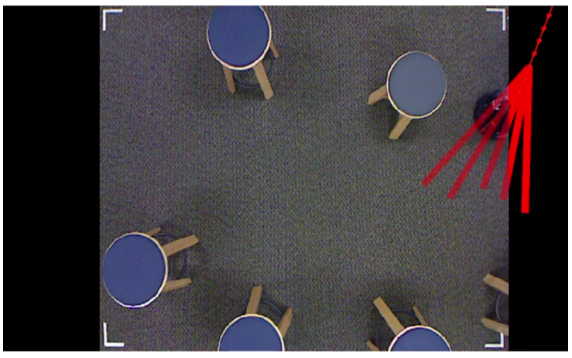
(b) Vehicle view 1-Navigation started



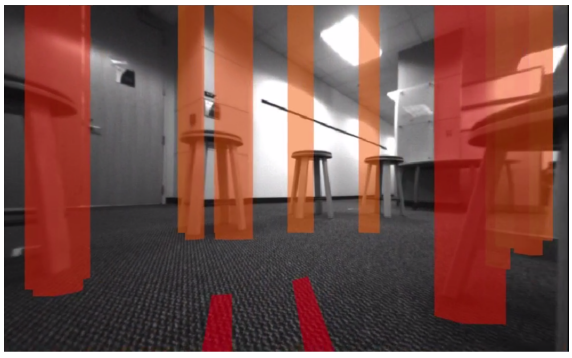
(c) Overhead view 2



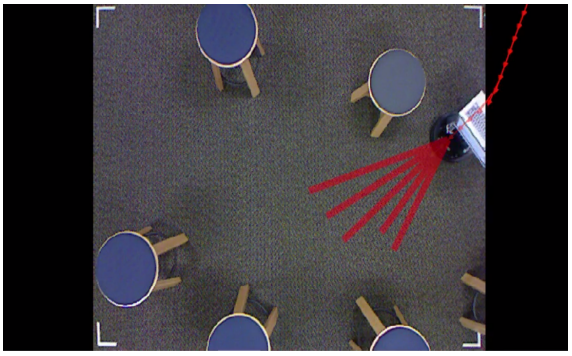
(d) Vehicle view 2



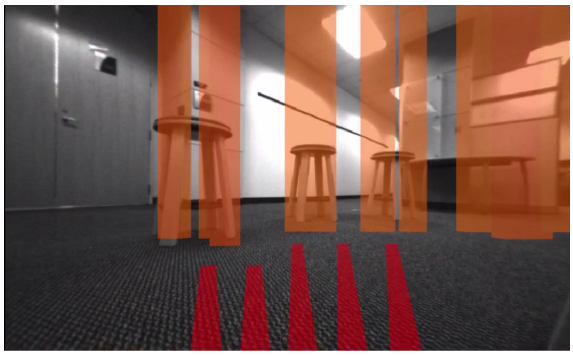
(e) Overhead view 3



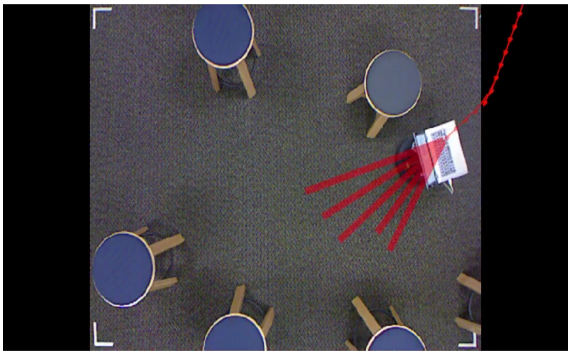
(f) Vehicle view 3



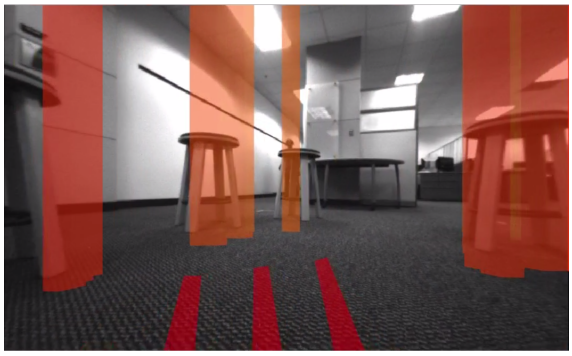
(a) Overhead view 4



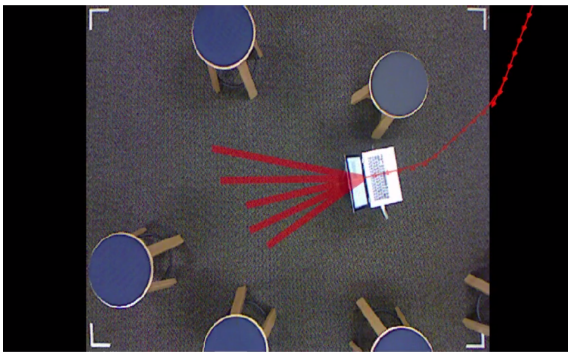
(b) Vehicle view 4



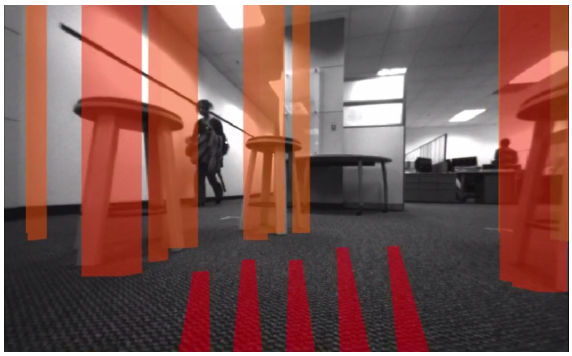
(c) Overhead view 5



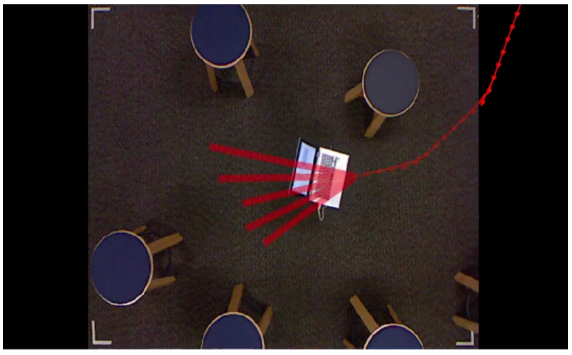
(d) Vehicle view 5



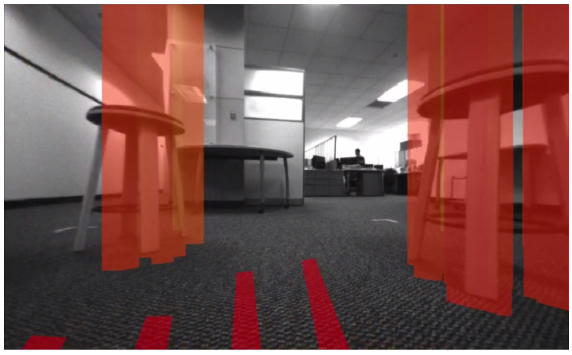
(e) Overhead view 6



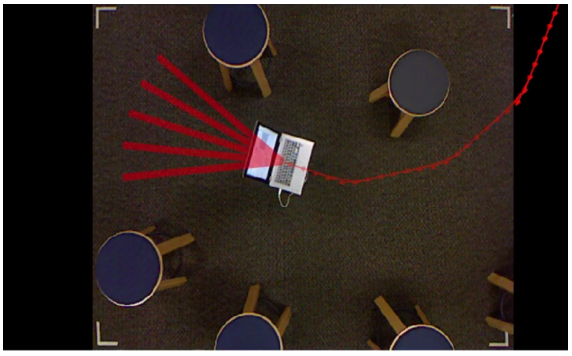
(f) Vehicle view 6



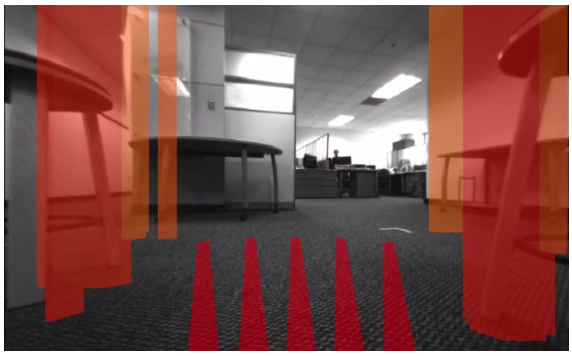
(g) Vehicle view 7



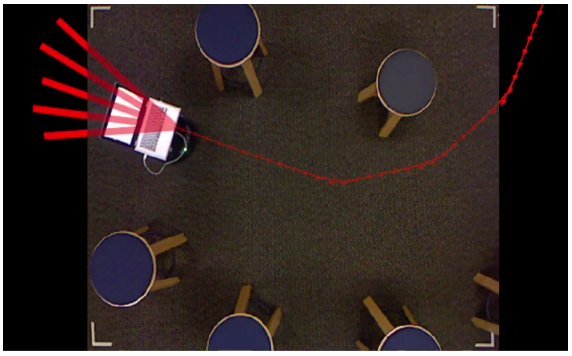
(h) Vehicle view 7



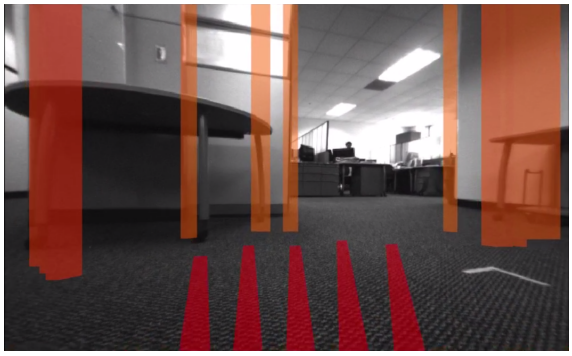
(a) Overhead view 8



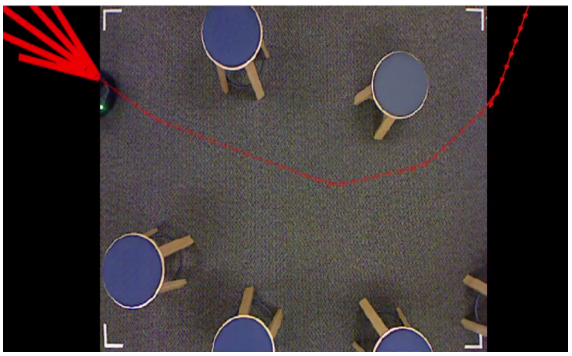
(b) Vehicle view 8



(c) Overhead view 9



(d) Vehicle view 9



(e) Overhead view 10-Navigation completed

(f) Vehicle view 10-Navigation completed

APPENDIX B

DERIVATION OF SCALAR DISTANCE FROM THE CAMERA ORIGIN TO THE INTERSECTION OF THE CIRCLE

We can express a position of a 3D Cartesian point as a product of a unit vector and a scalar distance.

$$\begin{aligned} x_b(u) &= \gamma_x \cdot t \\ y_b(u) &= \gamma_y \cdot t = y \\ z_b(u) &= \gamma_z \cdot t \end{aligned} \tag{B.1}$$

Constructed expression of a center radius expression of a circle is following:

$$(x_b - x)^2 + (z_b - z)^2 = r^2 \tag{B.2}$$

$$x_b^2 - 2x_b x + x^2 + z_b^2 - 2z_b z + z^2 = r^2 \tag{B.3}$$

$$(\gamma_x t)^2 - 2\gamma_x t x + x^2 + (\gamma_z t)^2 - 2\gamma_z t z + z^2 = r^2 \tag{B.4}$$

$$(\gamma_x^2 + \gamma_z^2)t^2 + (-2\gamma_x x - 2\gamma_z z)t + (x^2 + z^2 - r^2) = 0 \tag{B.5}$$

Using a quadratic formula, we express t as follows:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{B.6}$$

where

$$a = \gamma_x^2 + \gamma_z^2$$
$$b = -2\gamma_x x - 2\gamma_z z \tag{B.7}$$
$$c = x^2 + z^2 - r^2$$

Since we are only interested in the positive distance that goes forward from the camera origin. A complete expression is the following:

$$t = \frac{-(-2x\gamma_x - 2z\gamma_z) + \sqrt{(-2x\gamma_x - 2z\gamma_z)^2 - 4(\gamma_x^2 + \gamma_z^2)(x^2 + z^2 - r^2)}}{2(\gamma_x^2 + \gamma_z^2)} \tag{B.8}$$

REFERENCES

[1] D. Murray and J. J. Little, "Using real-time stereo vision for mobile robot navigation," *autonomous robots*, vol. 8, no. 2, pp. 161–171, 2000.

[2] F. Andert, "Drawing stereo disparity images into occupancy grids: measurement model and fast implementation," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 5191–5197.

[3] K. Konolige, E. Marder-Eppstein, and B. Marthi, "Navigation in hybrid metric-topological maps," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3041–3047.

[4] O. Veksler, Y. Boykov, and P. Mehrani, "Superpixels and supervoxels in an energy optimization framework," in *Proceedings of the 11th European Conference on Computer Vision: Part V*, ser. ECCV'10, Heraklion, Crete, Greece: Springer-Verlag, 2010, pp. 211–224, ISBN: 3-642-15554-5, 978-3-642-15554-3.

[5] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3 d shape descriptors," in *Symposium on geometry processing*, vol. 6, 2003, pp. 156–164.

[6] T. Cao, Z. Y. Xiang, and J. L. 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, 2015.

[7] 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)*, 2014, pp. 3242–3249.

[8] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[9] B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev, "Path planning for non-circular micro aerial vehicles in constrained environments," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3933–3940.

[10] C. Liu, "Safe robot navigation among moving and steady obstacles [bookshelf]," *IEEE Control Systems*, vol. 37, no. 1, pp. 123–125, 2017.

[11] J. Zhang, L. H. Wang, D. X. Li, and M. Zhang, "High quality depth maps from stereo matching and tof camera," in *2011 International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, 2011, pp. 68–72.

[12] R. Benenson, R. Timofte, and L. V. Gool, "Stixels estimation without depth map computation," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011, pp. 2010–2017.

[13] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 287–290, 2000.

[14] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[15] J. Sun, Y. Li, S. B. Kang, and H.-Y. Shum, "Symmetric stereo matching for occlusion handling," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, vol. 2, 2005, pp. 399–406.

[16] M. Perrollaz, J. D. Yoder, A. Spalanzani, and C. Laugier, "Using the disparity space to compute occupancy grids from stereo-vision," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 2721–2726.

[17] H. Badino, R. Mester, T. Vaudrey, U. Franke, and A. Daimler, "Stereo-based free space computation in complex traffic scenarios," in *Image Analysis and Interpretation, 2008. SSIAI 2008. IEEE Southwest Symposium on*, IEEE, 2008, pp. 189–192.

[18] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[19] P. Gohl, D. Honegger, S. Omari, M. Achtelik, M. Pollefeys, and R. Siegwart, "Omnidirectional visual obstacle detection using embedded fpga," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, pp. 3938–3943.

[20] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[21] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, IEEE, 1993, pp. 802–807.

[22] C. Rsmann, F. Hoffmann, and T. Bertram, "Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control," in *2015 European Control Conference (ECC)*, 2015, pp. 3352–3357.

[23] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H.-M. Tai, "Autonomous local path planning for a mobile robot using a genetic algorithm," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 2, 2004, 1338–1345 Vol.2.

[24] J. Sun, T. Mehta, D. Wooden, M. Powers, J. Rehg, T. Balch, and M. Egerstedt, "Learning from examples in unstructured, outdoor environments," *Journal of Field Robotics*, vol. 23, no. 11-12, pp. 1019–1036, 2006.

[25] L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 3242–3249.

[26] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, IEEE, vol. 3, 1997, pp. 2719–2726.

[27] G. E. Jan, K. Y. Chang, and I. Parberry, "Optimal path planning for mobile robot navigation," *IEEE/ASME Transactions on Mechatronics*, vol. 13, no. 4, pp. 451–460, 2008.

[28] J. Smith and P. Vela, "Planning in perception space," in *to appear IEEE International Conference on Robotics and Automation*, 2017.

[29] A. P. Moore, S. J. D. Prince, J. Warrell, U. Mohammed, and G. Jones, "Superpixel lattices," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[30] Z. Li and J. Chen, "Superpixel segmentation using linear spectral clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1356–1363.

[31] M. Schönbein and A. Geiger, "Omnidirectional 3d reconstruction in augmented manhattan worlds," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, IEEE, 2014, pp. 716–723.

[32] H. Badino, U. Franke, and D. Pfeiffer, "The stixel world-a compact medium level representation of the 3d-world.," in *DAGM-Symposium*, Springer, 2009, pp. 51–60.

[33] D. Pfeiffer and U. Franke, "Towards a global optimal multi-layer stixel representation of dense 3d data.," in *BMVC*, vol. 11, 2011, pp. 51–1.

[34] M. Cordts, L. Schneider, M. Enzweiler, U. Franke, and S. Roth, "Object-level priors for stixel generation," in *German Conference on Pattern Recognition*, Springer, 2014, pp. 172–183.

[35] L. Schneider, M. Cordts, T. Rehfeld, D. Pfeiffer, M. Enzweiler, U. Franke, M. Pollefeys, and S. Roth, "Semantic stixels: depth is not enough," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 110–117.

[36] D. Levi, N. Garnett, and E. Fetaya, "Stixelnet: a deep convolutional network for obstacle detection and road segmentation," in *Proceedings of the British Machine Vision Conference (BMVC)*, X. Xie, M. W. Jones, and G. K. L. Tam, Eds., BMVA Press, 2015, pp. 109.1–109.12, ISBN: 1-901725-53-7.

[37] M. Enzweiler, M. Hummel, D. Pfeiffer, and U. Franke, "Efficient stixel-based object recognition," in *2012 IEEE Intelligent Vehicles Symposium*, 2012, pp. 1066–1071.

[38] N. Bernini, M. Bertozzi, L. Castangia, M. Patander, and M. Sabbatelli, "Real-time obstacle detection using stereo vision for autonomous ground vehicles: a survey," *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 873–878, 2014.

[39] F. Erbs, B. Schwarz, and U. Franke, "From stixels to objects - a conditional random field based approach," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, 2013, pp. 586–591.

[40] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool, "Fast stixel computation for fast pedestrian detection," in *European Conference on Computer Vision*, Springer, 2012, pp. 11–20.

[41] B. Günyel, R. Benenson, R. Timofte, and L. Van Gool, "Stixels motion estimation without optical flow computation," in *European Conference on Computer Vision*, Springer, 2012, pp. 528–539.

[42] T. Scharwächter, M. Enzweiler, U. Franke, and S. Roth, "Stixmantics: A medium-level model for real-time semantic scene understanding," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, 2014, pp. 533–548.

[43] F. Erbs, B. Schwarz, and U. Franke, "Stixmentation-probabilistic stixel based traffic scene labeling.," in *Bmvc*, 2012, pp. 1–12.

[44] 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, 2005, 807–814 vol. 2.

[45] R. Labayrade, D. Aubert, and J.-P. Tarel, "Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation," in *Intelligent Vehicle Symposium, 2002. IEEE*, IEEE, vol. 2, 2002, pp. 646–651.

[46] S. Kubota, T. Nakano, and Y. Okamoto, "A global optimization algorithm for real-time on-board stereo obstacle detection systems," in *2007 IEEE Intelligent Vehicles Symposium*, 2007, pp. 7–12.

[47] J. A. Bondy, U. S. R. Murty, *et al.*, *Graph theory with applications*. Citeseer, 1976, vol. 290.

[48] J. Courbon, Y. Mezouar, and P. Martinet, "Autonomous navigation of vehicles from a visual memory using a generic camera model," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 3, pp. 392–402, 2009.

[49] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo matching using belief propagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 7, pp. 787–800, 2003.