# DROAN: Disparity space Representation for Obstacle AvoidaNce

Geetesh Dubey

CMU-RI-TR-17-42

July 2017

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Sebastian Scherer, Chair
Stephen Nuske
Sankalp Arora

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

**Keywords:** Collision Avoidance, Unmanned Aerial Systems, Micro Aerial Systems, Perception and Autonomy, Sensor Fusion, Obstacle Avoidance, Visual Navigation, Occupancy Mapping, Stereo Vision

*To my parents, sister and girlfriend.*

# Abstract

Agile Micro-Aerial-Vehicles or MAVs are required to operate in cluttered, unstructured environments at high speeds and low altitudes for efficient data gathering. Given the payload constraints and long range sensing requirements, cameras are the preferred sensing modality for MAVs.

However, contemporary approaches use stereo camera observations in 3D space by converting disparity image to point cloud and fail to deal with excess sensor noise at long ranges and often resort to using less noisy short range observations. The computation burden of using rich information provided by cameras and difficulty to deal with sensor error for obstacle sensing has forced the state of the art methods to construct world representations on a per frame basis, leading to myopic decision making.

In this thesis we propose a long range perception and planning approach using stereo cameras. We propose a method to use inverse-depth based obstacle representation which is adept at using the information provided by stereo cameras and enable to incorporate sensor noise model for probabilistic occupancy inference. Using 2D inverse-depth or disparity images based obstacle representation, our method enables computationally efficient, on-demand occupancy inference.

By utilizing FPGA hardware for disparity calculation and image space to represent obstacles, our approach and system design allows for construction of long term world representation whilst accounting for highly non-linear noise models in real time.

We demonstrate these obstacle avoidance capabilities on a quad-rotor flying through dense foliage at speeds of upto $10m/s$ for a total of $1.6$ hours of autonomous flights. The presented approach enables high speed navigation at low altitudes for MAVs for terrestrial scouting.

## Acknowledgments

I would like to thank Sankalp Arora for not only being a friend but a true mentor for life, research and career. I cannot thank enough Dr. Sebastian Scherer for the motivation, support and guidance he provided to me as an intern, a Research Associate and as a masters student. Thanks to Dr. Stephen Nuske for insightful discussions and providing me immense confidence during my research. I would also like to thank Dr. Sanjiv Singh for his faith in me and his immense support he provided to pursue my masters degree. Thanks to Greg Armstrong, Silvio Maeta, John Keller for helping with setting up and running experiments and fixing the robots.

This research is supported by the Office of Naval Research (Grant No. N00014-14-1-0693).

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Micro-aerial vehicles have long promised to be the agile sensing platforms of the future. MAV applications like stealth reconnaissance, search and rescue and cargo delivery etc. need fast aerial vehicles moving autonomously in cluttered environments, at low altitudes. Hence, fast and safe obstacle avoidance has remained an active research area. Achieving safe, autonomous, fast flight through cluttered environments on MAVs, presents two main challenges. One is the need for a large sensing horizon (Figure 1.1) to allow for adequate time to detect and avoid obstacles and second is the need for fast and accurate world representation update for minimal latency in reacting to newly discovered obstacles, (Figure 1.2).

Both of these challenges need to be addressed while keeping the sensing and computational payload to a minimum, to allow for maximizing MAV's flight time and agility. However, current state of the art systems that have demonstrated reliable autonomous flights in cluttered environments have either done so through active sensors like lidars [18], sacrificing agility and range of sensing, or have relied on monocular cameras and data driven techniques to provide proof of concept implementation in a specific environment [7]. Both schools of thoughts have led to pioneering demonstrations of obstacle avoidance capabilities of MAV's albeit at low speeds, either due to restricted sensing range or slow world/robot state updates.

Use of a stereo camera pair offers low weight, long range sensing at the cost of increased computation. Barry et. al [4] and Mathies et. al [16] have demonstrated obstacle avoidance at high speeds. While [16] perform motion planning on data observed in a single frame, [4] generate a relatively sparse map leading to a reactive behavior which is likely to be stuck in local minima in complex environments.

Integrating multiple disparity images, generated by a stereo pair is hindered by its highly non-linear noise model with noise monotonically increasing with distance. We propose an approach to integrate multiple disparity images in a computationally efficient manner to allow for long range, non-myopic obstacle avoidance using stereo data.

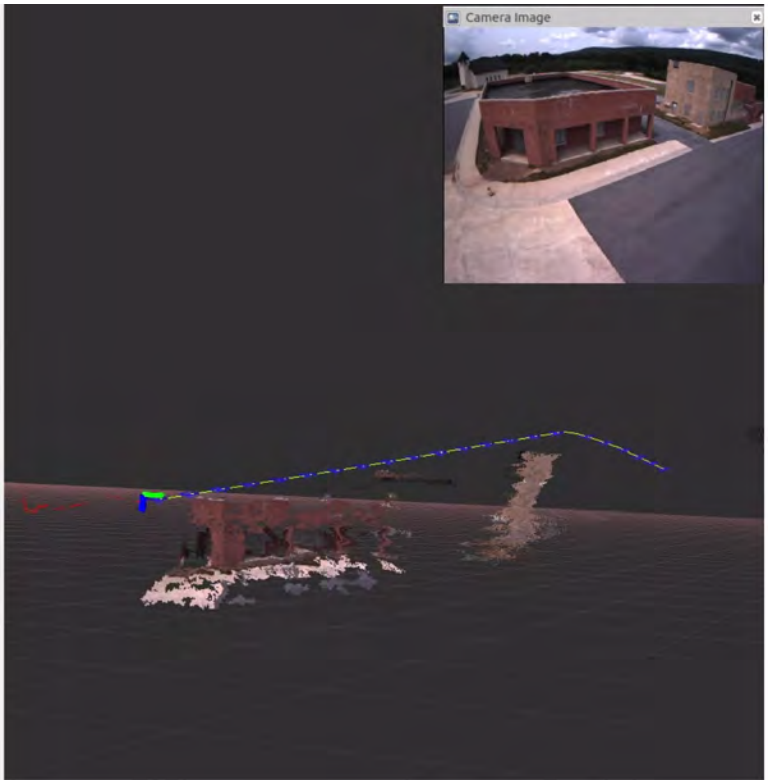



Figure 1.1: Long range perception and planning is needed to avoid obstacles at long distances. The observed building is approximately 40m away.

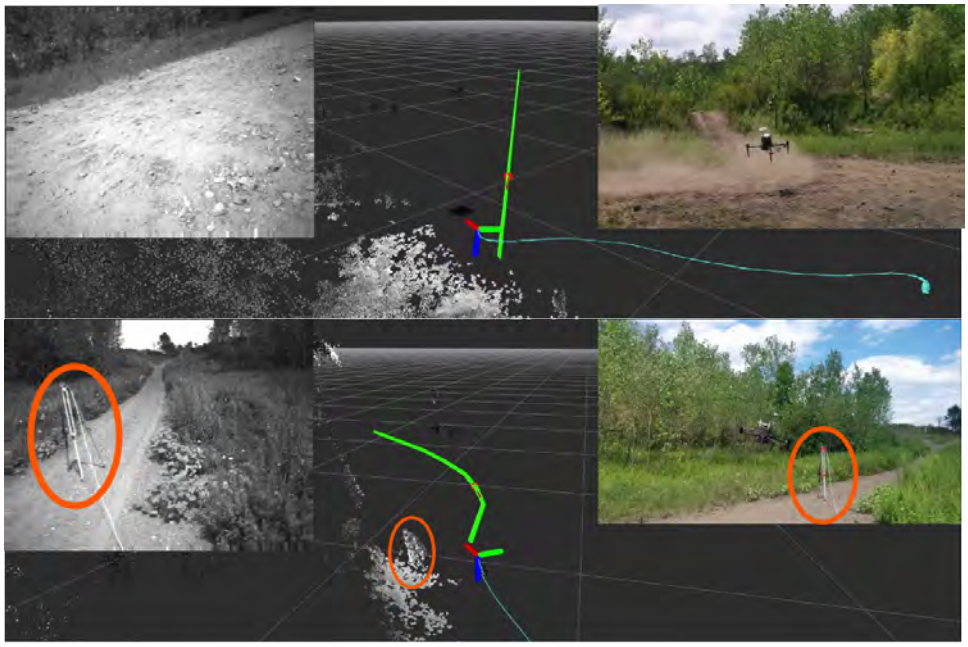


Figure 1.2: Low latency perception and planning is needed for low altitude high speed reactive navigation for MAVs. The robot makes a turn to go to the goal point and avoids the immediately seen obstacle. Obstacle is marked in red ellipse.

## 1.2 Contribution of the thesis

Our main contributions are -

- A principled application of disparity expansion as suggested by Mathies et. al [16] through incorporating a sensor error model and explicitly using two disparity images to create a front and back mask for obstacles.
- A principled approach to fuse information from multiple disparity images for probabilistic occupancy inference.
- System design and real-world demonstration of high speed ($10m/s$) obstacle avoidance in dense foliage, at heights as low as $1m$ AGL(above ground level).

## 1.3 Outline of the thesis

The thesis is structured as follows, Chapter 2 presents a short summary of the related work. Section 4.1 describes the overview of our planning algorithm. Chapter 3 describes various parts of the obstacle perception and Chapter 4 describes the planning algorithms in detail. Chapter 5 describes the MAV system on which the algorithm was tested with experimental results.

# Chapter 2

# Background

## 2.1 Related Work

Recently many research groups have started investigating vision-only obstacle detection and avoidance. Most approaches generate point clouds from disparity images and fuse with point clouds from other sensors such as lasers. Appropriate fusion of multimodal sensory data is still a work of active research. Most prevalent approach has been to generate 3D evidence grids or occupancy grids to determine occupancy and to check for collision [13], [1].

Working with 3D gridmaps is both memory intensive for large occupancy maps and requires more computation for registration of data and book keeping when scrolling or moving the grid along with the robot. Trade-off between high resolution gridmaps vs grid size is another reason why occupancy grids usually cannot be used to map a large volume with higher accuracy. OctoMaps [15] have recently become popular due to their efficient structure for occupancy mapping. However, due to excess noise in stereo sensor generated data at long ranges, often a smaller map is maintained and full stereo sensor data is not used.

Gohl et. al [10] proposes to use a spherical coordinate based gridmap suitable for stereo sensors. However every disparity map has to be converted to 3D point cloud before being injected into the 3D gridmap and hence it also suffers from the problem of computationally expensive step of conversion to 3D point cloud and subsequent map warping or scrolling as the robot moves.

A pushbroom stereo scanning method is proposed in [4] for obstacle detection for MAVs flying at high speeds. As the robot moves, disparity measurements equal to a fixed value are collected to generate a map of the environment. This allows significant speed-up in map building as a disparity is only generated if there is a correspondence in right image at predefined disparity from the left image. Since the collected disparity measurements are at a fixed distance, usually not very far to obtain reliable measurements from stereo camera, it is only suitable for short distance detection and reactive planning.

We base our work on [16] which proposed a Configuration-Space (C-Space) expansion step to apply an extra padding around disparities based on robot size. The method in [16] works when planning in spaces where the stereo system disparity is not very noisy i.e. in close range making the planning system myopic in nature and prevents long planning horizon. Another drawback of this approach is that it only concerns with closest detections and disregards any information

available beyond closer detections.

Although cameras provide rich information about the environment, the aforementioned approaches fall short of utilizing this rich information and there is a need to find an obstacle representation that is both, suited to vision sensing modality and efficient to process the rich data.

# Chapter 3

# Disparity Space Perception

As discussed in the previous chapter, there is a need to find an obstacle representation that is both, suited to vision sensing modality and efficient to process the rich information we propose a 2D inverse-depth or disparity image based representation. Since we represent obstacles as a set of images itself, it serves as the closest data structure to a camera image. Also, 2D representation is quick for inference as lookup is a simple pixel access. This chapter discusses in depth and reasons about why this approach is suitable. We also reason about why there is a need to maintain a local map and propose a solution to maintain a spatial memory as the robot navigates through the environment.

## 3.1 Local Perception & Planning

We use disparity image or inverse depth image for obstacle representation as it naturally captures spatial volume according to the sensor resolution [10]. This representation is befitting for noisy stereo data as explained in Section 3.1.1. We employ C-space expansion where the original disparity image is expanded, allowing us to treat the robot as a point when doing collision checks during planning [16].

Our method incorporates a stereo sensor error model and allows us to reason about space behind obstacles. We use an additional padding in disparity both in front and behind obstacles. This padding varies from $3\sigma$ for close obstacles to $1\sigma$ for far obstacles, where $\sigma$ is the standard deviation of disparity error and the multiplier is represented by $\lambda$ in later sections. By varying $\lambda$ we ensure safe planning at short range and a more optimistic planning at long range. This enables the deliberative planning required for exploration tasks.

### 3.1.1 Disparity error and its effects

Disparity is a measure of the proximity of an obstacle. We can derive how close the obstacle is in depth using triangulation in stereo vision as follows.

$$z = \frac{bf}{d} \tag{3.1}$$

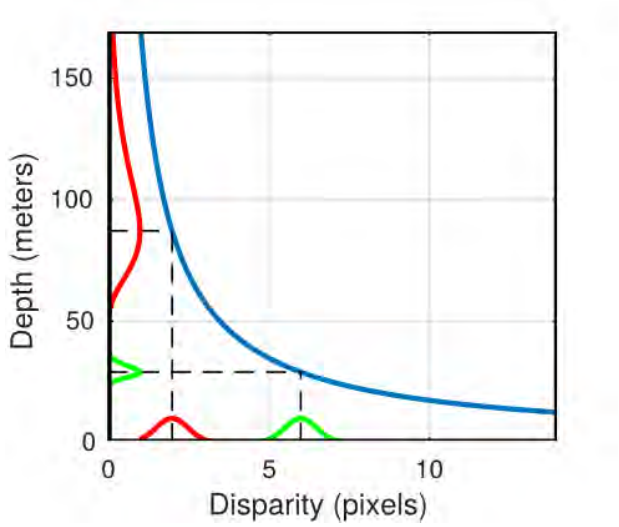



Figure 3.1: Disparity vs Depth (blue) and probability distributions are shown in red and green. Red and Green PDF in disparity are same and easy to model but their corresponding Red and Green PDF in range vary and difficult to model. Hence we use inverse depth space to represent obstacles. Also, disparity i.e. inverse range captures space at multi-resolution suitable for registration of stereo sensor data.

Where, $z$ is the depth of a pixel$(u, v)$ with disparity $d$, $b$ is baseline and $f$ is the focal length in pixels.

The actual 3D point can be derived as

$$P(x, y, z) = (uz/f, vz/f, z) \tag{3.2}$$

The accuracy of the stereo setup is drastically affected as the disparity decreases. The error in depth increases quadratically with depth as shown in equation(3.5). Differentiating equation(3.1) wrt $d$

$$\frac{\partial z}{\partial d} = -\frac{bf}{d^2} \tag{3.3}$$

$$\partial z = -\frac{z^2}{bf}\partial d \tag{3.4}$$

$$\partial z \sim z^2 \tag{3.5}$$

Disparity error is primarily caused due to correspondence error while matching pixels along the epipolar line. It can be modelled using a Gaussian pdf. Assuming correspondence error during disparity computation has a std deviation $\sigma = 0.5pixels$, we define the Gaussian pdf $\mathcal{N}(d, \sigma^2)$. Figure 3.1 shows how this Gaussian pdf in disparity results in a difficult to model pdf for error in depth with an elongated tail on one side and a compressed tail on the other. This motivates to use disparity image space domain directly for occupancy inference rather than resorting to depth or 3D domain. In next Section3.1.2 we empirically determine the sensor error and derive the inverse sensor model.

### 3.1.2 Estimating disparity error

In stereo vision the stereo camera geometry is used to compute the inverse depth or disparity to a seen object in left and right image. There is an error when computing this disparity in pixels given some disparity computation algorithm. We want to model this error in disparity by setting up an experiment.

We want to find a probability distribution of disparity error given a disparity image and some ground truth. To this end we generate disparity using the FPGA based solution [17] and the ground truth is obtained by placing a chequered/chess board pattern in front of the sensor. Using the metric information of the corners on the board and by detecting those corners in the image we can obtain the pose of each corner with respect to the camera which can be used as ground truth. Corresponding to each corner pixel the disparity is sampled. We collect several such samples and generate a histogram of disparity error. Figure 3.2 shows the setup at a distance of 2.5m. Data was sampled at various other distances as discussed in the next section.

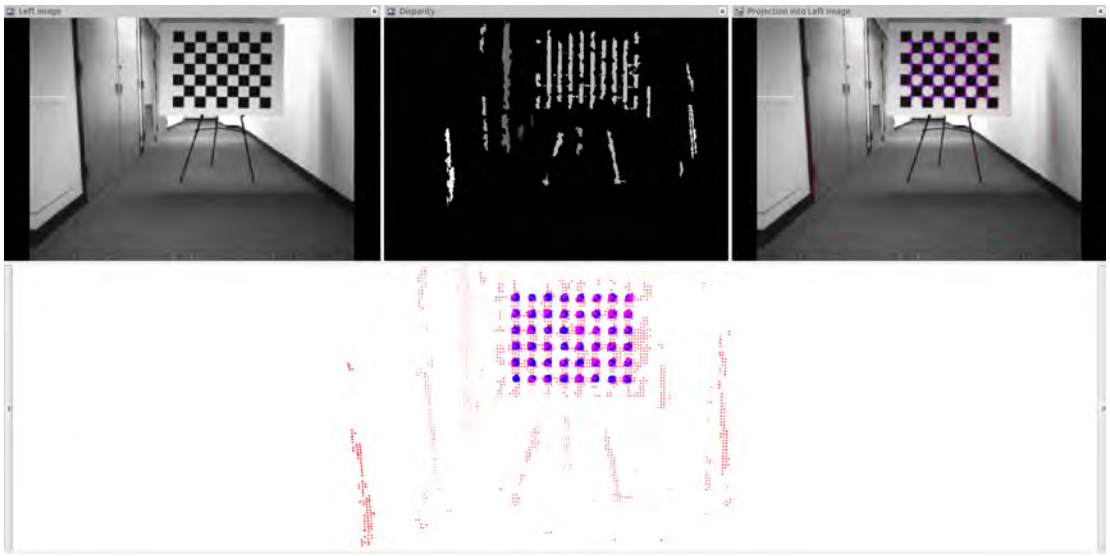
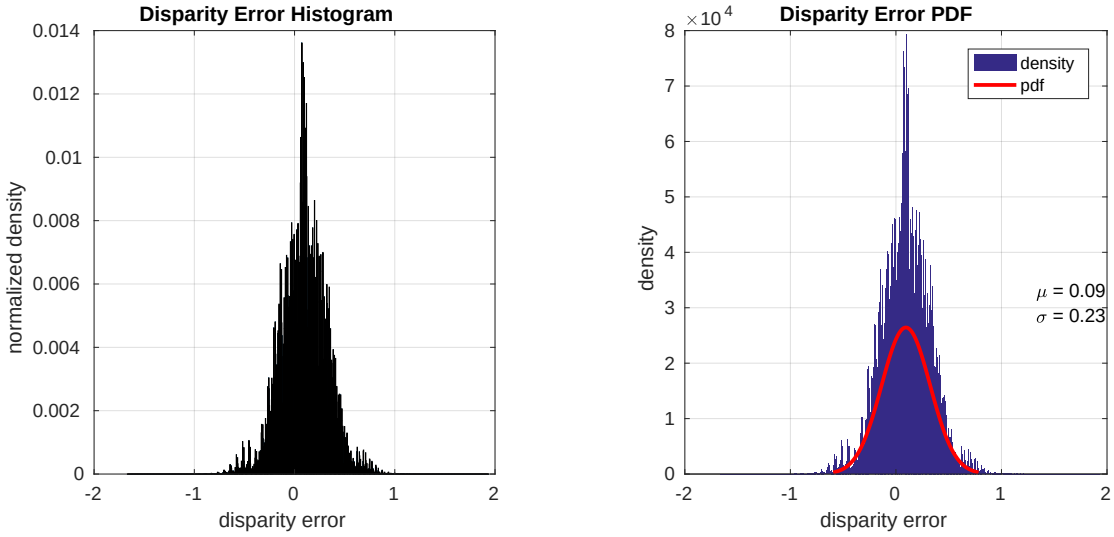


Figure 3.2: Experiment setup to collect data for error modelling. The point cloud at bottom shows the ground truth points in blue and sampled points in pink. The small sized red point cloud is the disparity generated point cloud.

### 3.1.3 Disparity error modelling

Based on the above experiments we obtained following results:

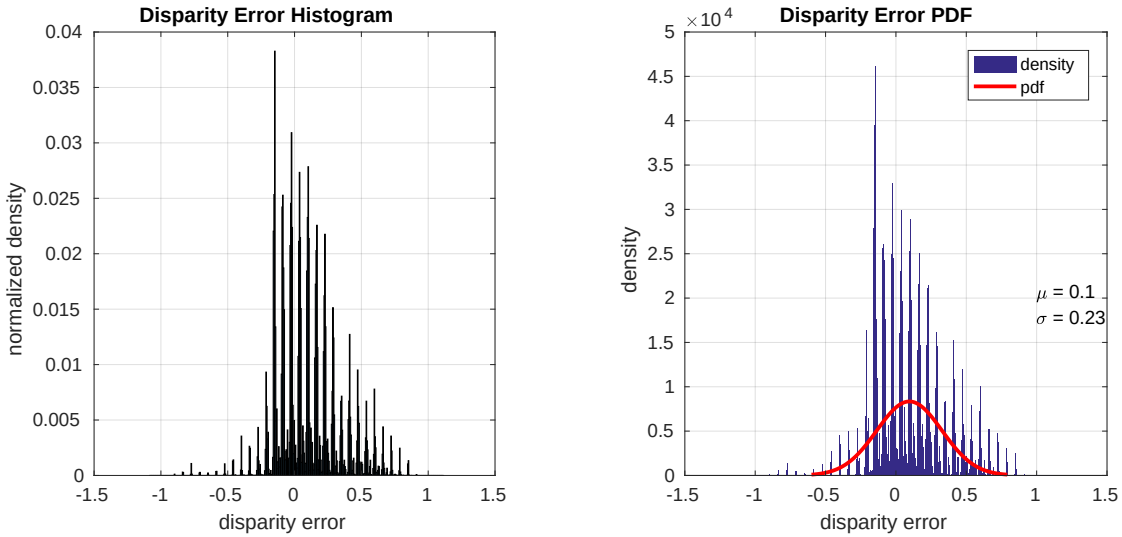



(a) Disparity error histogram

(b) PDF for disparity error

Figure 3.3: Disparity Error profile at $2.5m$ from stereo sensor.

Figure 3.3 shows the disparity error histogram on left and the fitted Gaussian pdf on normalized histogram. It can be seen that a simple Gaussian fits well with a standard deviation $\sigma = 0.23$ pixels at a depth of $2m$.

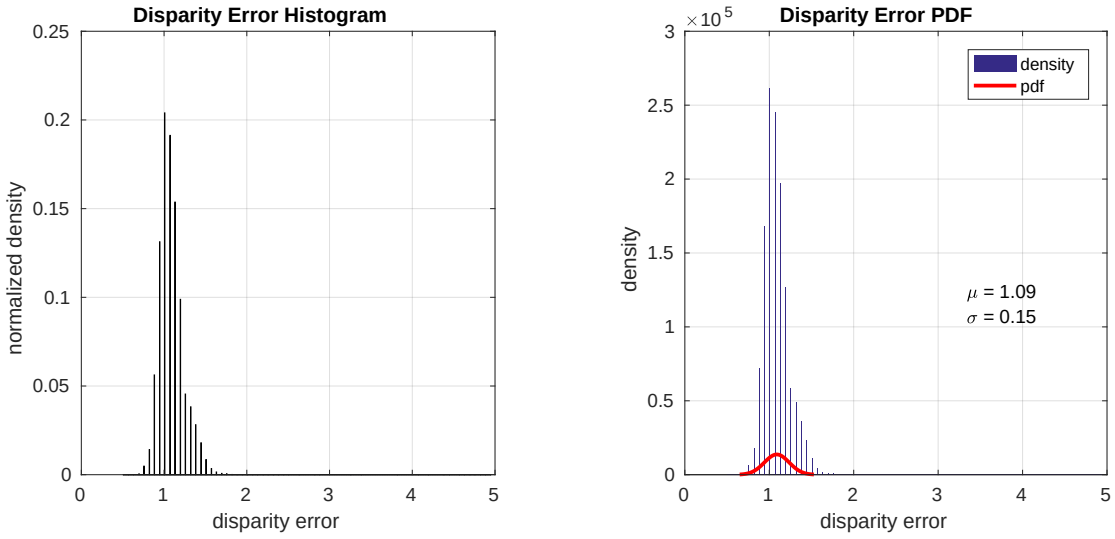



(a) Disparity error histogram

(b) PDF for disparity error

Figure 3.4: Disparity Error profile at $3.8m$ from stereo sensor.

Figure 3.4 shows similar results but at approximately twice the depth i.e. at $4m$. It also fits the Gaussian pdf with same standard deviation. This means in disparity space the fitted Gaussian does not change or the change is not significant as we will see in upcoming results.

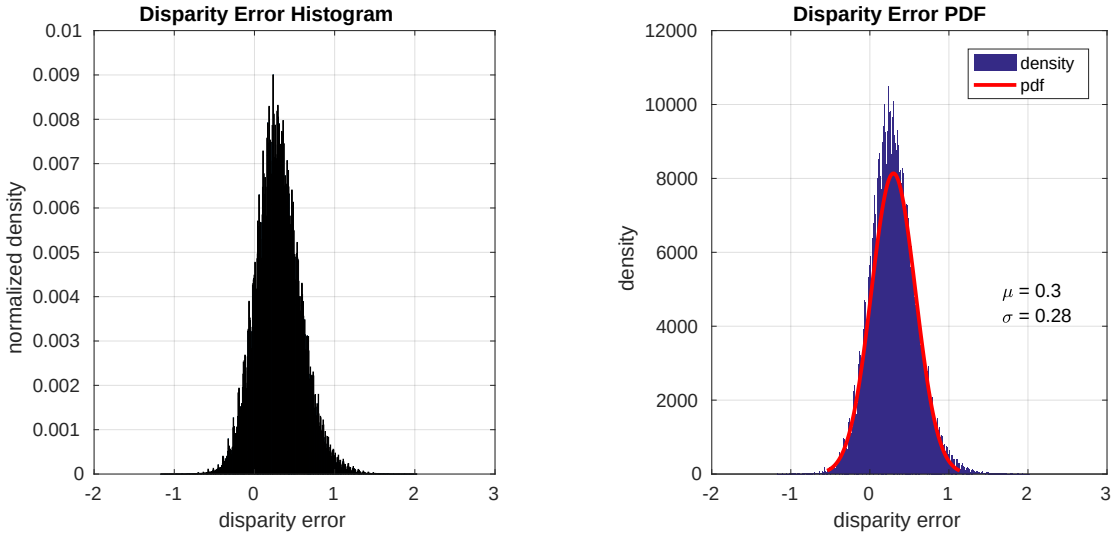



(a) Disparity error histogram

(b) PDF for disparity error

Figure 3.5: Disparity Error profile at $10.0m$ from stereo sensor.

Figure 3.5 shows the histogram of disparity error and the fitted Gaussian distribution. Here we see some departure from previous results but this can be attributed mostly to human error because the ground truth had to be collected manually as the pattern was too far to be detected automatically by the chessboard detector.




(a) Disparity error histogram

(b) PDF for disparity error

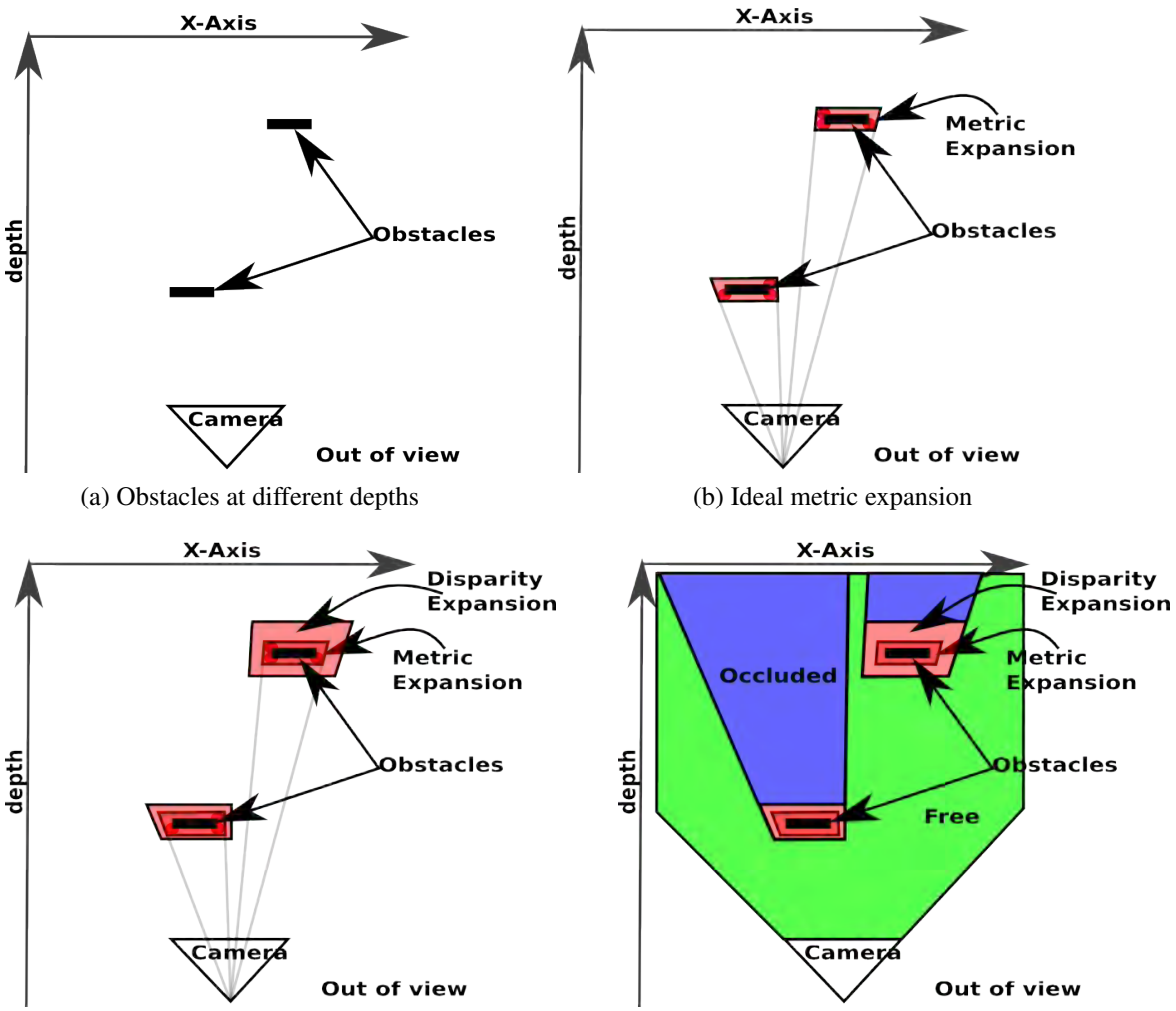
Figure 3.6: Disparity Error profile produced by moving stereo sensor about $2.0m$ away from the chessboard pattern.

Figure 3.6 shows the results with the pattern being moved around in front of the sensor. This also shows similar results to the first two experiments.

**Note:** In all distributions there is a constant mean of error and this can be attributed to discrepancies in the chess board itself. It was difficult to keep the board rigid and not flex. This must lead to the error in computation of the ground truth itself.

After conducting these experiments we obtain the $\sigma$ of disparity error that can be used for the Gaussian sensor error model.

### 3.1.4 Configuration-Space expansion




(a) Obstacles at different depths

(b) Ideal metric expansion

(c) Disparity expansion with sensor noise

(d) Expanded C-Space

Figure 3.7: C-Space expansion illustration: (a) Shows two obstacles at different depths and (b) shows their metric expansion using a fixed robot radius. (c) shows expansion after incorporating stereo sensor error model. (d) illustrates how different areas are inferred as occupied, free and unknown or occluded.

Configuration-Space is the domain in which collision checking for planning tasks is done. Usually for 3D navigation C-Space can be a 3D gridmap. C-Space expansion is required to represent obstacles such that a single point state query can be used for collision checks. Occupancy grids have been the default methods for registration of sensor data and C-Space expansion for occupancy inference. Usually point clouds are used to populate occupancy grids but point cloud generated using disparity images are highly uncertain at greater depths and hence occupancy grid based representation is infeasible. Moreover, 3D occupancy grids require a huge amount of memory to capture the planning workspace and hence fail to incorporate long range measurements available from stereo sensors. To overcome this limitation we use disparity images and

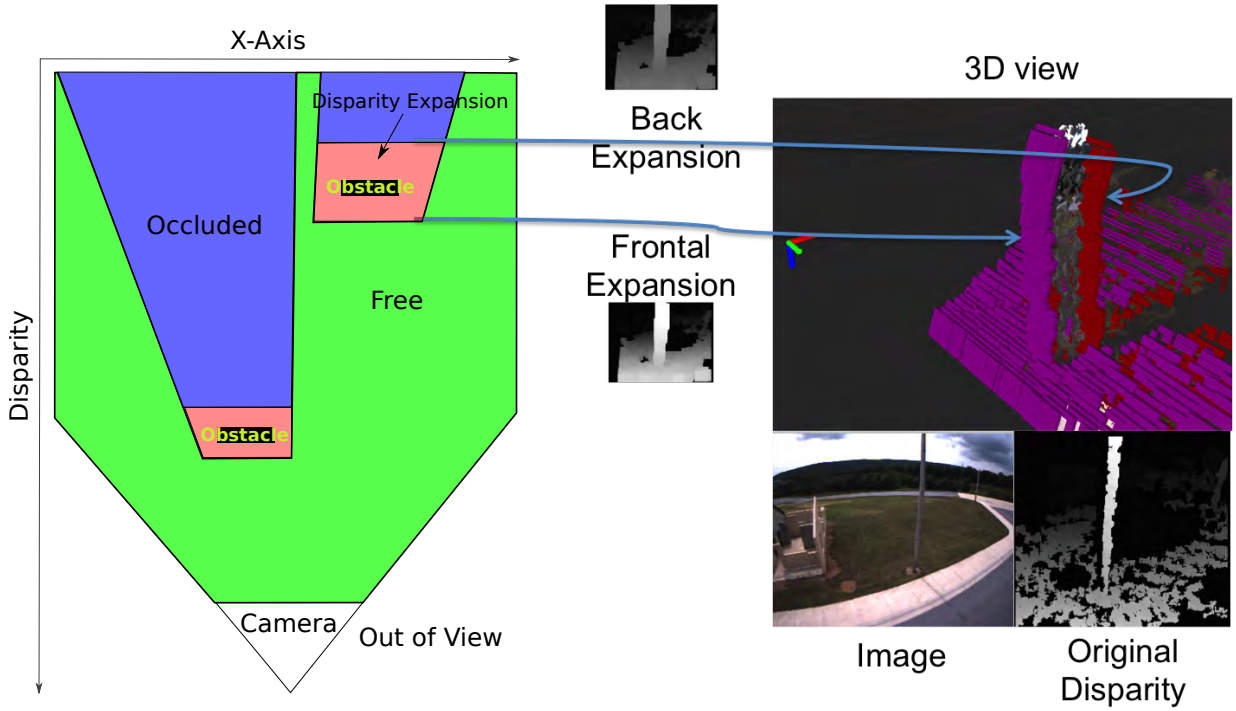



Figure 3.8: Disparity expansion shown as point cloud. The pink and red point cloud represent the foreground and background disparity limits.

apply disparity expansion step. Figure 3.7 shows a 2D illustration of what is meant by disparity expansion. Figure 3.7(a) shows two obstacles at different depths. Figure 3.7(b) shows how an ideal metric expansion would look like given the robot radius for expansion. The dark red circles represent the robot. Figure 3.7(c) shows hows this expansion is affected if there is sensor measurement noise. Figure 3.7(d) shows the final expanded C-Space that will be used for planning. The algorithm is explained in section 3.1.5.

### 3.1.5 Disparity expansion

In this section we explain the step of C-Space expansion as applied to disparity images. This step allows us to capture the volume occupied by an obstacle using two surfaces represented by two disparity images. These images represent front and back surface limits of the reported disparity. Each pixel in these two images effectively captures the range of disparity based on robot size and the sensor error model as shown in the Figure 3.1. This process can be divided into two steps.

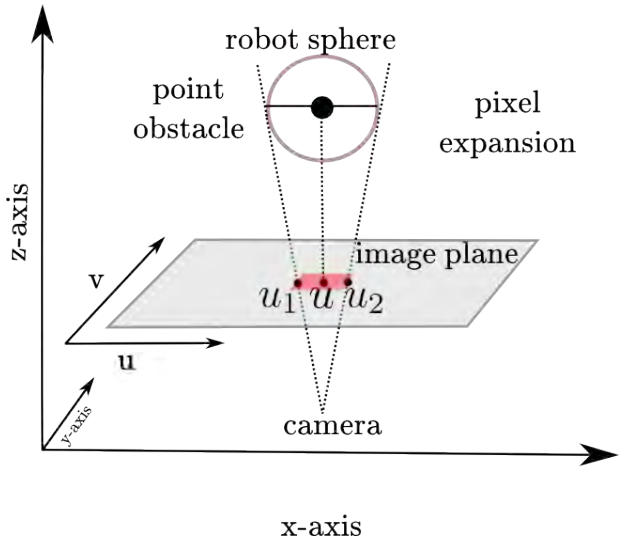



Figure 3.9: Shows the pixel-wise expansion of a point obstacle according to robot size.

The first step expands disparities along the image $XY$ axis Figure 3.9 i.e. an obstacle at some pixel $(u, v)$ after inflation occupies a manifold of pixels from $[u_1, u_2]$ and $[v_1, v_2]$. This is achieved by traversing through the image row-wise first and then column-wise. This is similar to [16] but we also incorporate sensor error. We omit the steps required to generate the look-up-table (LUT) to map $u \rightarrow [u_1, u_2]$ given disparity $d$ and $v \rightarrow [v_1, v_2]$ given disparity $d$. Reader is advised to refer [16] for generation of the LUT, but unlike looking up for the raw disparity value $d$ from table we look up for $(d + \lambda\sigma)$, where $\lambda$ is the sigma multiplier dependent on the range as discussed previously in Section 3.1.

The second step expands disparities to get new values for front and back images using equation(3.6). These images represent the maximum and minimum disparities for every pixel respectively.

$$\begin{aligned} z &= \frac{bf}{d} \\ d_{\mathrm{f}} &= \frac{bf}{z - r_v} + \lambda\sigma \\ d_{\mathrm{b}} &= \frac{bf}{z + r_v} - \lambda\sigma \end{aligned} \tag{3.6}$$

Where $r_v$ is the expansion radius based on robot size, $d_{\mathrm{f}}$ and $d_{\mathrm{b}}$ are the computed front and back disparities which encompass the obstacle. As shown in illustration on left side of Figure 3.8, the red area around the original disparity of obstacle is the padding generated in the expansion step. This padding is based on the robot size and sensor error model.

Our approach uses the LUT as shown in Algorithm(1) which takes the original disparity image $D$ as input and processes it to generate the expanded frontal and back disparity images $D_f$ and $D_g$ respectively. The function $expand(d)$ implements equation(3.6) with $\lambda = 0$ and $r_v = 0$ to prevent double expansion in depth. The function $connectedComponent()$ searches for minimum disparity connected to the maximum disparity over steps of provided $range$ (set to

a multiple of robot radius). This helps to find an obstacle bounding volume. We do not want to use the minimum disparity in a window as that can be located very far with no connection to the actual obstacle and hence the $connectedComponent()$ step is required.

**Algorithm 1:** Disparity Expansion Algorithm

**Input**: Disparity image $D$
**Output**: Expanded disparity images: $D_f, D_b$
**for** $v = 1 : Height(D)$ **do**
  **for** $u = 1 : Width(D)$ **do**
    $\hat{d} = ceil(D(u,v) + \lambda\sigma)$
    $[u_1, u_2] = LUT(u, \hat{d})$
    $V = D(u_1 : u_2, v)$ `// Get vector of disparities`
    $d_f = expand(max(V))$
    $d_b = expand(connectedComponent(d_f, range))$
    **for** $i = u_1 : u_2$ **do**
      $D_f(i,v) = \max(d_f, D_f(i,v))\ D_b(i,v) = \min(d_b, D_b(i,v))$
    **end**
  **end**
**end**

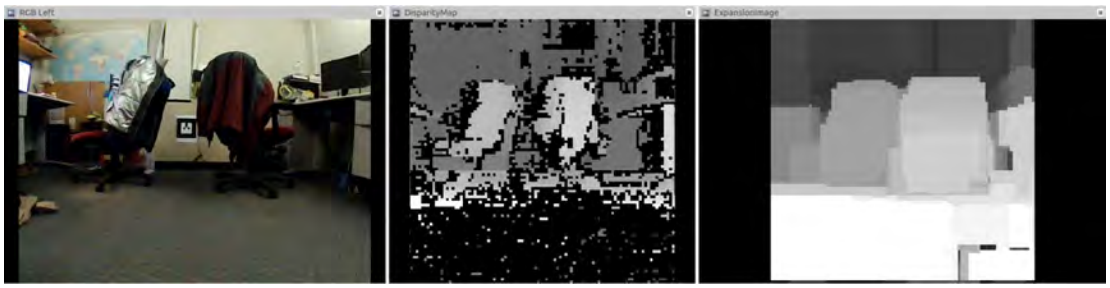
Algorithm(1) does row-wise expansion and its result is then subject to column-wise expansion in a similar fashion with $\lambda$ and $r_v$ set to default values when using $expand()$ function. The expanded disparity images constitute a single snapshot volumes occupied by obstacles. Figure 3.10 shows some examples of expansion result.

To maintain a spatial memory we create a pose graph consisting of multiple expanded disparity images as described in the following section.

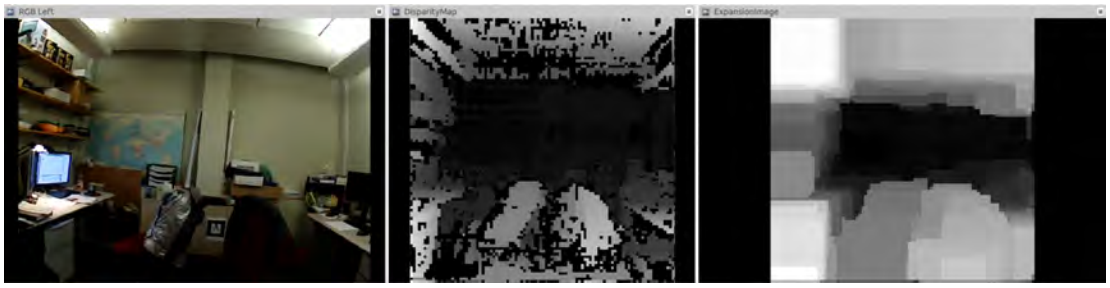
### 3.1.6 Pose graph of disparity images

A single observation is often not enough to construct a reliable occupancy map, hence several observations are fused into a local map enabling local spatial memory. Moreover, stereo cameras only observe the environment in the overlapping field of view. Hence, a spatial memory is required to create a local map of the environment as the robot moves in it. We propose to use a pose-graph of our disparity image based representation to maintain a spatial memory. A pose-graph can further benefit from a simultaneous localization and mapping solution to correctly register the observations and later use to generate a global occupancy map. In this thesis we are not concerned with generation of a global map but it should be noted that it is possible using this approach. The motivation to maintain spatial memory of the previously seen environment as the vehicle is moving using a pose graph is because of the following reasons:

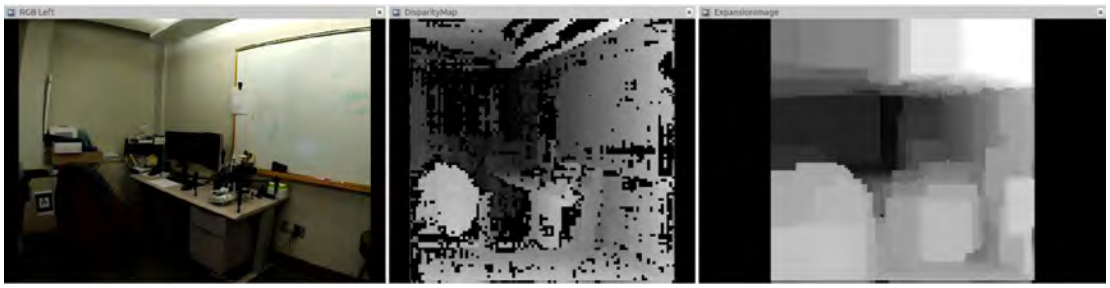
1. Previously seen obstacles might not be visible in the current image.
   (a) The stereo sensor has a minimum range dependent on maximum perceivable disparity.
   (b) Obstacles get occluded in different views.
   (c) The field of view is limited.
2. Maintain a pose graph of disparity images (measurements) with nodes at regular intervals of distances and angles as shown in Figure 3.11.
3. Allows occupancy inference using multiple measurements.




(a)




(b)




(c)

Figure 3.10: Left to right: Original Image, disparity around cropped part of image, Frontal Expansion result.

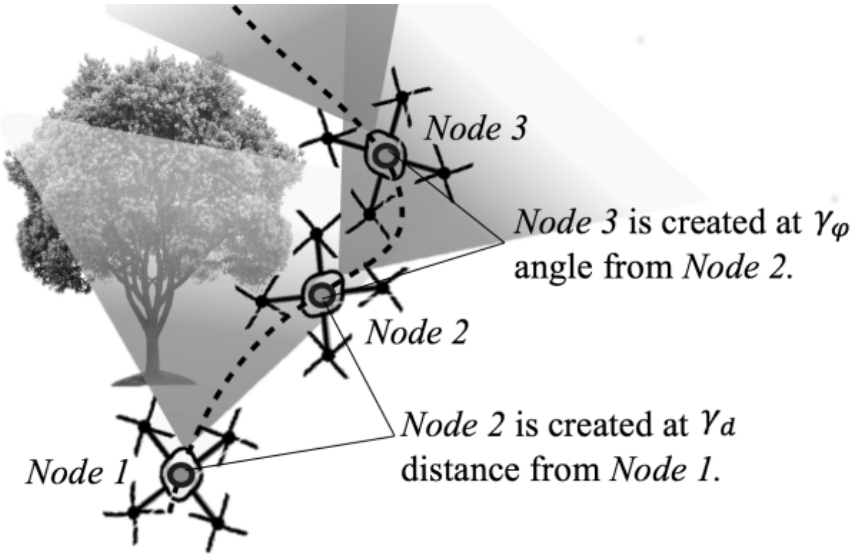



Figure 3.11: Pose Graph of expanded disparity images. Dashed path shows robot motion and stored nodes in the graph are shown as triangles. Nodes are stored at intervals of distance and orientation.

Algorithm(2) shows how we construct this graph. Each node in the graph is comprised of

**Algorithm 2:** Pose Graph Algorithm

**Input**: $D_f, D_g, Pose, N_{graph}, \gamma_d, \gamma_\psi$
**Output**: Pose Graph of Expanded disparity images: $Graph$
$T_s^w \leftarrow Pose$
$Node = createNode(T_s^w, D_f, D_b)$
**if** $Graph.size() == 0$ **then**
  $Graph.push\_front(Node)$
  $Graph.push\_back(Node)$
**end**
$PrevNode = Graph.begin()$
$pos\_err = distance(Pose, PrevNode)$
$ang\_err = angle(Pose, PrevNode)$
**if** $pos\_err >= \gamma_d || ang\_err >= \gamma_\psi$ **then**
  **if** $Graph.size() == N_{graph}$ **then**
    $Graph.pop\_back()$
  **end**
  $Graph.push\_front(Node)$
**end**
$Graph.pop\_back()$
$Graph.push\_back(Node)$

the following:

1. $D_f$
2. $D_b$
3. $T_s^w$ which is the transform between the processed sensor measurement($D_f, D_b$) and world frame.

The algorithm takes as input the current robot position $Pose$, processed disparity images $D_f, D_b$, maximum number of nodes $N_{graph}$ and two tolerance parameters $\gamma_d, \gamma_\psi$ for position and angular displacement respectively. The constructed graph is used to project a given world point into all node images and do occupancy inference. Occupancy inference using the set of disparity

images in the graph is explained in subsequent section.

### 3.1.7 Occupancy inference

Occupancy inference is the method to derive occupancy of a volume using all the observations or evidence we have. This is a widely studied topic and is often not a trivial problem. [20] Chapter-9 explains this in detail.

Evidence grids or occupancy maps are the practical methods to allow fusion of different measurements taken over time and build an occupancy grid map. Occupancy in an evidence grid is determined using log odds update. Each cell in the grid map represents the summation of log odds of the probability of occupancy given the measurements $z_1, ..., z_t$:

$$odds(m_{x,y,z}|z_{1:t}) = \frac{p(m_{x,y,z}|z_{1:t})}{1 - p(m_{x,y,z}|z_{1:t})} \tag{3.7}$$

where $m_{x,y,z}$ is a binary variable indicating whether the cell with coordinates $(x, y, z)$ is occupied and $p(m_{x,y,z}|z)$ is the probability of occupancy based on the inverse sensor model. By applying Bayes rule and using Markov assumption this simplifies to an additive update when using log-odds.

$$L(m_{x,y,z}|z_t) = L_{t-1} + log\frac{p(m_{x,y,z}|z_t)}{1 - p(m_{x,y,z}|z_t)} \tag{3.8}$$

By maintaining a pose graph of expanded disparity images, we can also take advantage of similar fusion without building an occupancy grid which are not suited for stereo data as discussed previously. We devised an occupancy inference method by fusing information from all the images in the graph using the stereo sensor error model. Next section will discuss this in more detail.

### 3.1.8 Probabilistic inference in disparity space

We want to formulate a probabilistic inverse sensor model for the stereo sensor used. To this end let us define some desirables from such a model.

1. **Probability Mass:** We want the sensor model to return a probability of obstacle presence over a range of disparity that represents the volume of the robot in disparity space.
2. **Optimistic in occluded space:** We want the model to to be more optimistic about occupancy in occluded areas to allow for deliberative planning as in exploration planning scenarios.

We use the Gaussian sensor error model as explained earlier and make the following analysis. Figure3.12 shows how the occupied volume changes in disparity space given a fixed size robot.

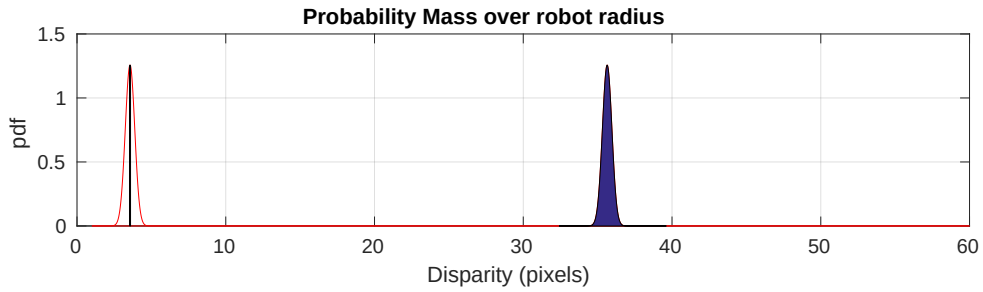



Figure 3.12: Probability mass (shown in blue area) occupied by a robot of radius $0.5m$ at a distance of $50m(3.5px)$ and $5m(35.6px)$. As the distance increases or disparity decreases the probability mass occupied by the robot varies.

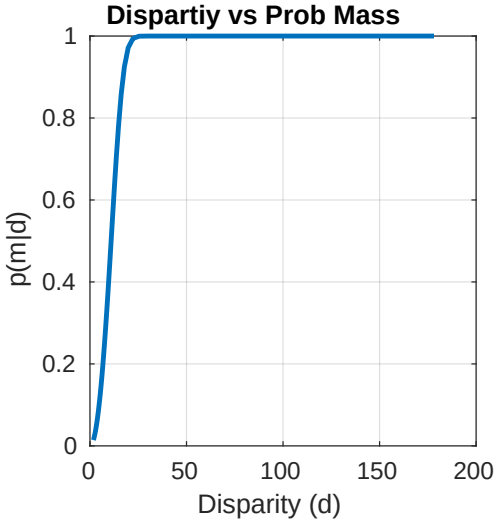



Figure 3.13: Probability Mass of occupancy, given a robot at disparity $d$ pixels. This curve is used to compute the log-odds to infer occupancy.

Based on this, Figure 3.13 shows the probability mass as a function of inverse depth or disparity. It is clear that for high disparity or low range the probability of occupancy given a measurement is very high and drops quickly after a certain disparity. This is because for the same Gaussian distribution at different disparities the actual range of disparity that the robot would occupy falls drastically.

However, it is computationally expensive to obtain such a probability using a CDF function online. Hence, we want to find a function that closely resembles this curve. In next section we introduce a confidence function which we use online for occupancy inference instead.

### 3.1.9 Confidence function for inference in disparity space

For occupancy inference using log-odds we need an inverse-sensor-model. Moreover, for a stereo camera sensor and a given robot radius we need to be able to compute probability mass given the inverse-sensor-model. It is difficult to compute this probability mass online as it requires integration of inverse-sensor-model pdf over the robot radius. Although we can pre-compute this probability mass and use approximation such as piece-wise linear interpolant as a function of disparity but we propose a new confidence function which is inexpensive to compute online. Given the standard deviation of correspondence error $\sigma$, we compute confidence of a disparity state in the following manner.

$$C(d) = \frac{(d - \sigma)}{d} \tag{3.9}$$

Confidence measure from equation(3.9) gives us a measure of how much can we trust a given disparity for occupancy inference.

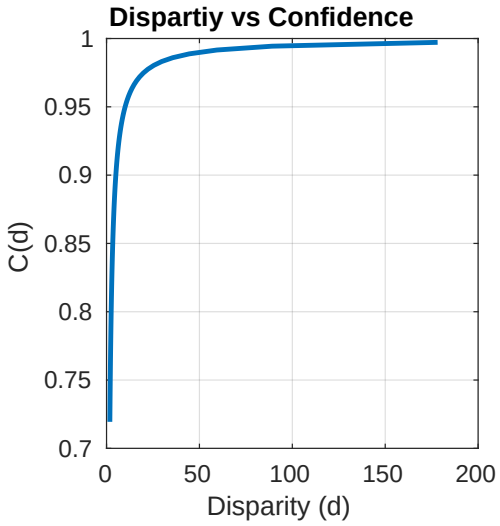



Figure 3.14: Disparity vs Confidence plot obtained from equation (3.9). This is inexpensive to compute online compared to computation of probability mass which involves integration of inverse-sensor-model over robot radius.

Figure(3.14) shows how this measure relates to disparity.

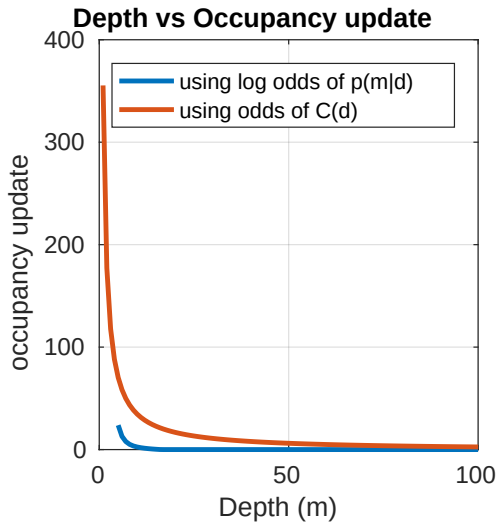



Figure 3.15: Occupancy update comparison between log-odds and proposed confidence inference. Confidence based occupancy update is more conservative in nature and will lead to detection of an obstacle using fewer observations. Hence, this function is pessimistic about free-space and ensures an obstacle will be detected in all cases when log-odds also detects an obstacle.

Since we are interested in occupancy inference we compare the occupancy update using the log-odds probabilistic inference and the proposed confidence inference method. Figure(3.15) shows the plot of occupancy update using the two methods.

The confidence function is more conservative in nature when doing occupancy update at longer ranges. Hence, it can be said if the probabilistic inference would update a state as occupied the proposed confidence function will also mark it as occupied. It can be seen that at long range or low disparity, uncertain measurements have low confidence and update the occupancy with lower values. We further discount measurements that mark an area safe or potentially safe(occluded) by 0.5 to be more conservative about clearing areas previously marked occupied. It should be noted that the potentially safe areas are behind obstacles and have lower disparity state, hence their contribution to occupancy clearance is less due to lower confidence value. In our experiments we get the final occupancy measure by projecting a world point $P$ using equation(3.10) and equation(3.2) in disparity images of all nodes in the graph and accumulating the occupancy cost according to Table(3.1):

### 3.1.10 Collision checking

Collision checking is used to plan a new path and to validate if an existing path is safe to follow. Collision checking is performed using the following mapping of a 3D world point $P$ to image pixel $I$ with disparity $d_s$:

$$P(x, y, z) \leftrightarrow I(u, v, d_s) \tag{3.10}$$

A state is in collision if the occupancy measure as shown in equation(3.11) crosses a pre-

Table 3.1: Occupancy update

| Check | Remark | occupancy cost $occ(d_s)$ |
|---|---|---|
| $d_s > d_{\mathrm{f}}(u,v)$ | safe | $-0.5\frac{C(d_s)}{1-C(d_s)}$ |
| $d_s < d_{\mathrm{f}}(u,v)$ and $d_s > d_{\mathrm{b}}(u,v)$ | obstacle | $\frac{C(d_s)}{1-C(d_s)}$ |
| $d_s < d_{\mathrm{b}}(u,v)$ | potentially safe | $-0.5\frac{C(d_s)}{1-C(d_s)}$ |

defined threshold $\gamma$.

$$Occupancy = \sum_{nodes} occ(d_s) \tag{3.11}$$

$$Occupancy \geq 0.0 \tag{3.12}$$

If the occupancy for a state is below the threshold, we consider that state as not occupied by an obstacle.

# Chapter 4

# Planning

## 4.1 Planning Pipeline

Mathies et. al [16] presented an approach to use disparity images generated by a stereo pair for obstacle avoidance. In this approach the occupied pixels in the disparity image obtained from the stereo pair are expanded to account for robot's size. The expanded disparity images are used as a spatial representation to plan collision free paths.

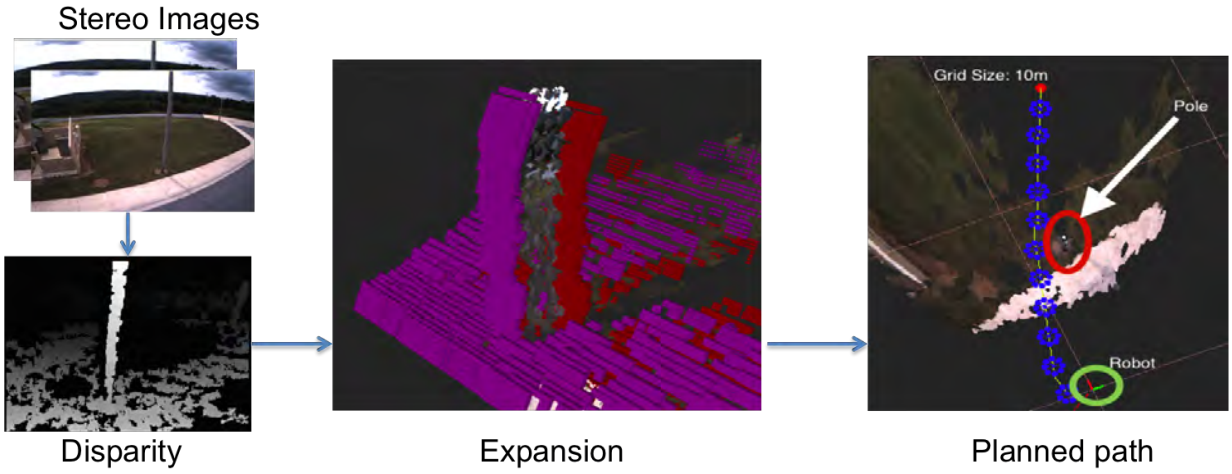



Figure 4.1: Planning pipeline based on inverse depth obstacle perception. The frontal expansion and back expansion are shown in pink and red point cloud around the original point cloud of pole. Planned path around the pole is also shown with the current robot position circled in green.

We use the a similar algorithm to expand the disparity images but adapt the expansion step through the inclusion of the observation noise model in the disparity expansion which is more suited for real-world stereo-sensor returned data. Furthermore, we compute two image expansions; frontal and back to probabilistically capture the occupancy region. Figure 4.1 shows how the two images capture the pole obstacle. The frontal expansion is shown in pink point cloud and the back expansion in red. We also improve the path planning by using multiple disparity images to infer occupied volumes. The use of multiple disparity images allows the planner to

reason about long range obstacles. The improved expansion algorithm and multi-image occupancy inference are presented in section 3.1.10. Furthermore, all the planned paths end at hover position with zero velocities ensuring safety of the vehicle. Figure 4.1 briefly shows the planning pipeline.

We explored two types on planners both with their own advantages and disadvantages.

1. Sampling based Planner:
2. Trajectory Library based Planner:

Both are discussed in the next sections.

## 4.2 Sampling Based Planner

We first chose to use a sampling based planner as such a planner finds a plan from start position to goal position. Using such a planner we can examine how the disparity space representation of obstacles holds for long range end-to-end planning.

We use a sampling based planner, BIT* [9] to draw samples in 3D space which are checked for collision as described in 3.1.10. The output is a collision free path connecting start to goal state.

In our experiments we found that disparity images fluctuate around obstacle edges leading to unwanted replanning due to the current plan being in collision. To remedy this we used two threshold values. A lower value $\gamma_{low}$ is used during planning to find a path i.e. obstacles are observed sooner even at long distances and hence a more conservative path is obtained. A higher threshold value $\gamma_{high}$ is used to check the current plan for collision and do replanning in case of collision. The advantage of using two threshold values is that an initial plan is found using a more conservative occupancy map while the replanning is done using a more reliable occupancy map. The reliable occupancy map is not affected by fluctuations in the disparity maps. The thresholds are chosen such that collisions at close range are always detected but have great advantage to not force replanning due to less reliable and fluctuating observations at long range when planning paths to longer distances. In our experiments we have planned paths at distances longer than $100m$ (Figure 5.10). Figure 4.1 shows a planned path that avoids a pole obstacle. This path is sent to the motion controller of the vehicle.

In environments which are densely populated with obstacles, sometimes it can take too long to find a smooth path connecting the start state to goal state. This can become a critical aspect when trying to avoid obstacles reactively and simultaneously finding a full path to a goal located far from robot or start position. So, we also tested the disparity-space representation for collision checking using a trajectory library based planner as discussed in the next section.

## 4.3 Trajectory Library Based Planner

Trajectory or manoeuvres libraries have been widely used in the robotics community to solve high dimensional control problems such as grap selection or for trajectory set generation for mobile robot navigation [8],[19],[5],[11],[6]. The motivation to use a prior set of libraries is that they effectively discretize a large control or planning space and enable good performance within

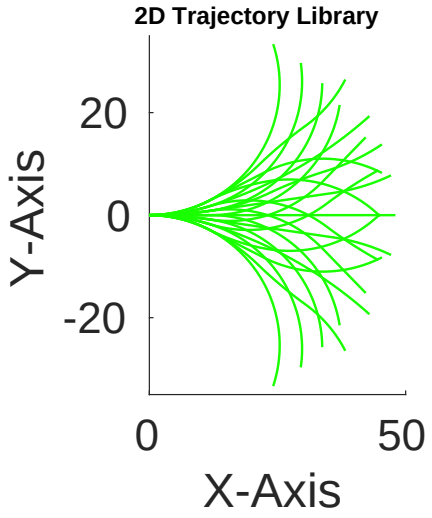



Figure 4.2: 2D Trajectory Library


Figure 4.3: Example of 3D Trajectory Library used on the robot. The Axes represent the robot, red is forward x-axis and blue is down z-axis.

possible computational limits. All candidates in the library are evaluated at runtime and one is chosen if success is achieved or some cost function is minimized. However, the performance is hugely affected by the size and content quality/coverage of the library. Size refers to the number of candidates that can be evaluated during runtime and quality or coverage refers to dispersion of the candidates [12]. The main advantage of using such libraries is that they are guaranteed to be dynamically feasible and hence allow smooth motion or manipulation.

## 4.4 Motion Control

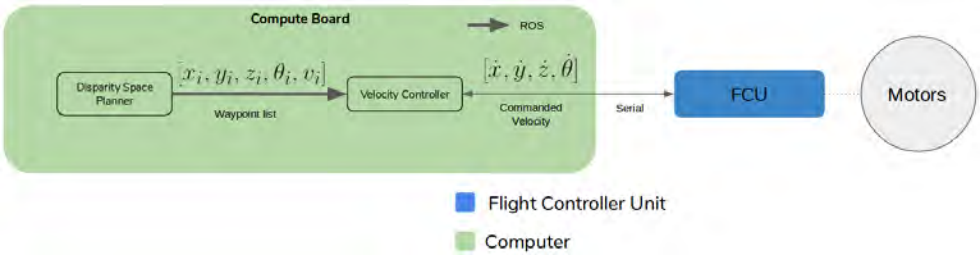



Figure 4.4: Motion control block diagram. The planner sends out a list of waypoints consisting of 3D positions, heading and a desired velocity to the velocity controller. The velocity controller generates a new velocity command to be able to track the path with the desired velocity and sends it out to the flight control unit(FCU). The FCU on our system is capable of receiving velocity commands and does the final task of generating necessary motor commands.

We developed a path tracker similar to [14]. It takes the current trajectory and uses feed-forward velocities specified in the trajectory and generates final velocity and heading rates for the low level velocity controller. The low level velocity controller runs on the quadrotor's flight control unit as shown in Figure 4.4.

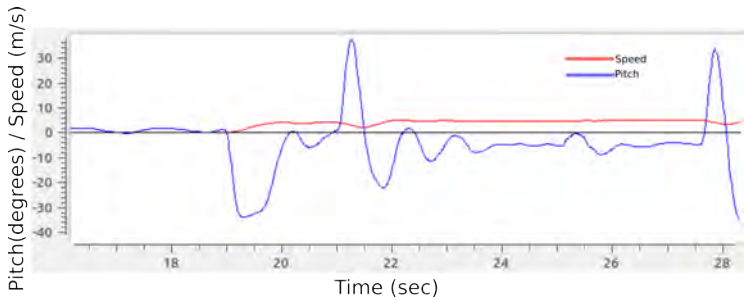



Figure 4.5: Vehicle Pitch when accelerating to a speed of $4m/s$ during one of the trials. The vehicle levels out in pitch quickly with a low offset from horizontal as the desired speed can be reached soon. This allows the cameras pitched $15°$ down to observe the space in front of the robot for discovery of obstacles.

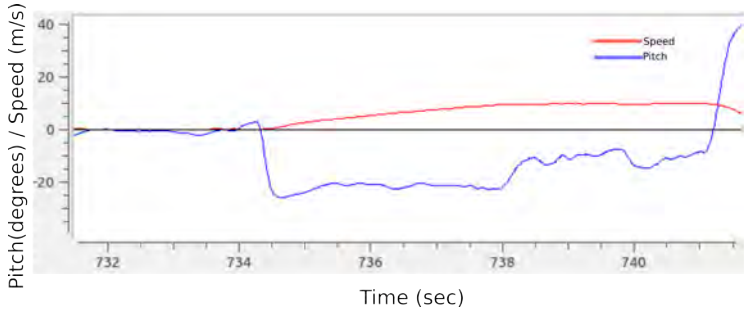



Figure 4.6: Vehicle Pitch when accelerating to a speed of $10m/s$ during one of the trials. The vehicle takes a long time to level in pitch and there is a higher offset in pitch from the horizontal after reaching the desired speed. This often leads to very late discovery of obstacles as the cameras are unable to see directly in front in the direction of motion.

Figure 4.5 and Figure 4.6 show how the vehicle pitch behaves as the desired speed is changed. The plots show data from real obstacle avoidance runs. It can be noted there is a higher constant offset from level pitch angle as the desired speed increases. On our system as explained in 5.1 the stereo cameras are pitched down at an angle of $15^{\circ}$ and this lead to difficulties in detection of obstacles at long range in $10m/s$ obstacle avoidance runs.

Next section provides further details on how the motion controller works.

### 4.4.1 Trajectory tracking

A path, $P \in N \times R^3$ , is defined by a sequence of N desired waypoints, $x_i^d$ and desired speeds of travel $v_i^d$ along path segment $P_i$ connecting waypoint $i$ to $i+1$, as depicted in Figure4.7. Let $t_i$ be the unit tangent vector in the direction of travel along the track from $x_i^d$ to $x_{i+1}^d$ , and $n_i$ be the unit normal vector to the track. Then, given the current position $x(t)$, the cross track error $e_{ct}$ and along track error rate $\dot{e}_{at}$ are,

$$
\begin{aligned}
e_{ct} &= (x_i^d - x(t)) \cdot n_i \\
\dot{e}_{ct} &= -v(t) \cdot n_i \\
\dot{e}_{at} &= v_i^d - v(t) \cdot t_i
\end{aligned}
$$

Only along track error rate is used to track the path so that the resulting controller does not attempt to catch up or slow down for scheduled waypoints,but simply proceeds along the track matching the desired velocity as closely as possible. A trajectory tracking controller was implemented by closing the loop on along track velocity and cross track error.

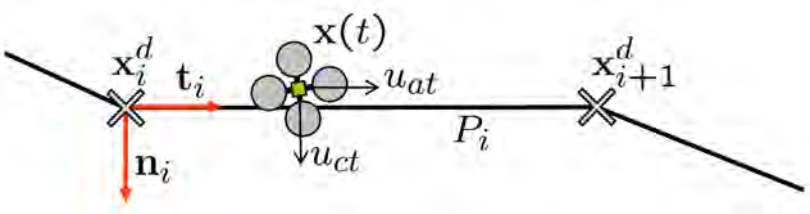



Figure 4.7: Vehicle path definition. While the quadrotor travels along segment $P_i$ from waypoint $x_i^d$ to $x_{i+1}^d$ , it applies along track and cross track control inputs $u_{at}$ and $u_{ct}$ to follow the path segment [14].

The cross track acceleration $a_{ct,i}$ at waypoint $x_i = (xi, yi)$ is a function of the velocity $v_{ct,i}$ at $x_i$ and the radius of curvature $r_i$ ,

$$a_{ct,i} = \frac{v_i^2}{r_i}$$

Constraining the cross track acceleration to be of magnitude less than $a_{max}$ results in a maximum allowable velocity $v_i$,allow at $x_i$ ,

$$v_{i,allow} \leq \sqrt{a_{max} r_i}$$

This way a speed limit is imposed between every waypoint.

# Chapter 5

# Results

## 5.1 System

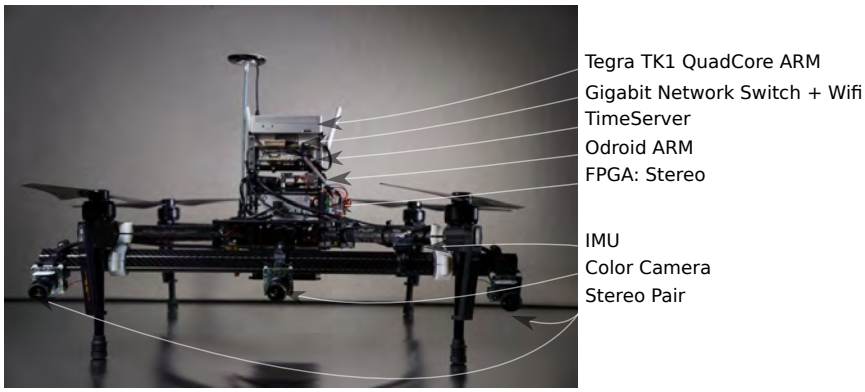



Figure 5.1: Quadrotor platform used for experiments: equipped with stereo camera sensor suite and onboard ARM computer

We test our algorithms on-board an autonomous UAV system, see Figure 5.1. The base platform is an off-the-shelf quadrotor vehicle retrofitted with in-house developed sensing and computing suite designed for semantic exploration. The sensor suite consists of a monochrome stereo camera pair, a monocular color camera, an integrated GPS/INS unit and a barometer. The stereo camera pair provides $640 \times 480$ resolution disparity image at $10$ fps for the obstacle avoidance and 3D mapping systems. The central camera is operated at a lower frame rate, to provide high resolution color imagery for the semantic perception system. All cameras are forward-facing, tilted downwards at $15^{\circ}$, an orientation well suited for low-altitude ($< 40m$) operation. The GPS/INS system and the barometer are used for state estimation.

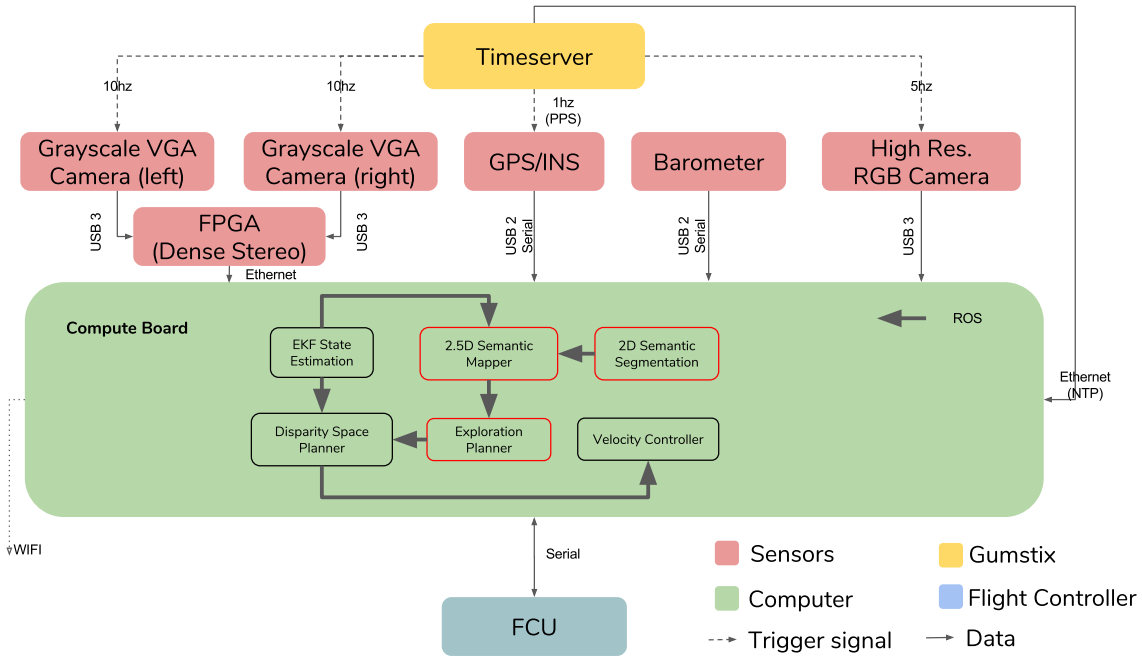



Figure 5.2: The system architecture diagram shows the hardware and software components and data exchange. The software components with red border are not my contributions as a part of this thesis.

Figure 5.2 shows the high level system diagram and architecture. All computation for autonomous operation is performed onboard. To this end we equip the MAV with two embedded ARM computers; one of them is devoted primarily to planning tasks, while the other is devoted to perceptual tasks. In addition, we use a specialized FPGA processor [17] for stereo depth computation. The computers are networked through high-speed ethernet.

## 5.2 Visualization: Obstacle grid map generated using inverse-depth representation

Following are some visualizations of the perception algorithm and generated plans. Figure 5.3 to Figure 5.6 show the projected plan (green path) and projected occupancy map (height colored voxels) in the camera on top left, current disparity image on top right and 3D visualizations on bottom with first person view on left and top down view on right. The blue arrow/axes shows robot pose, green arrows show the keyframe pose in the disparity graph and the RGB ball shows the goal. The small points represent the noisy point cloud generated using the disparity image colored by height. The voxel grid map is produced by uniformly sampling a $100 \times 100 \times 20m^3$ volume centred at robot with $0.5m$ resolution. For clarity voxels close to ground have been removed. This is only for visualization purpose and does not need to be produced for onboard planning.

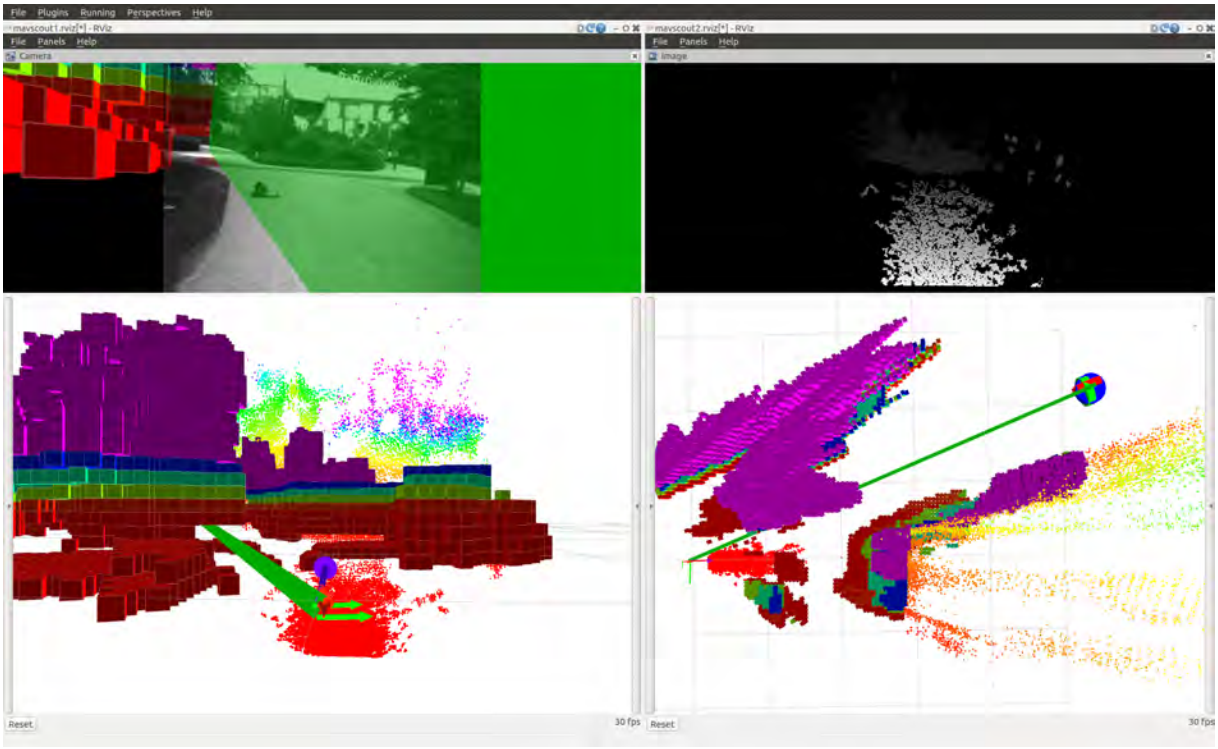


Figure 5.3: Straight path under a tree.

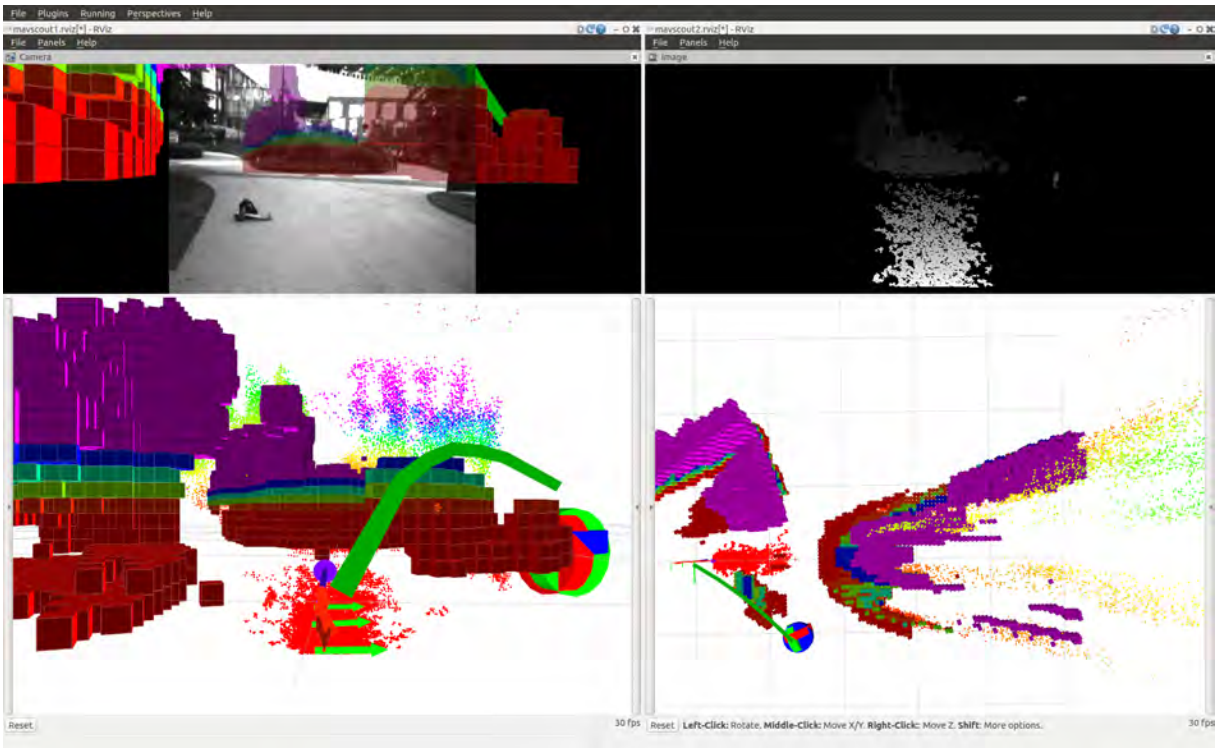


Figure 5.4: Planned path to a location not in current view. The spatial memory using disparity graph allows to plan to locations not in current view.

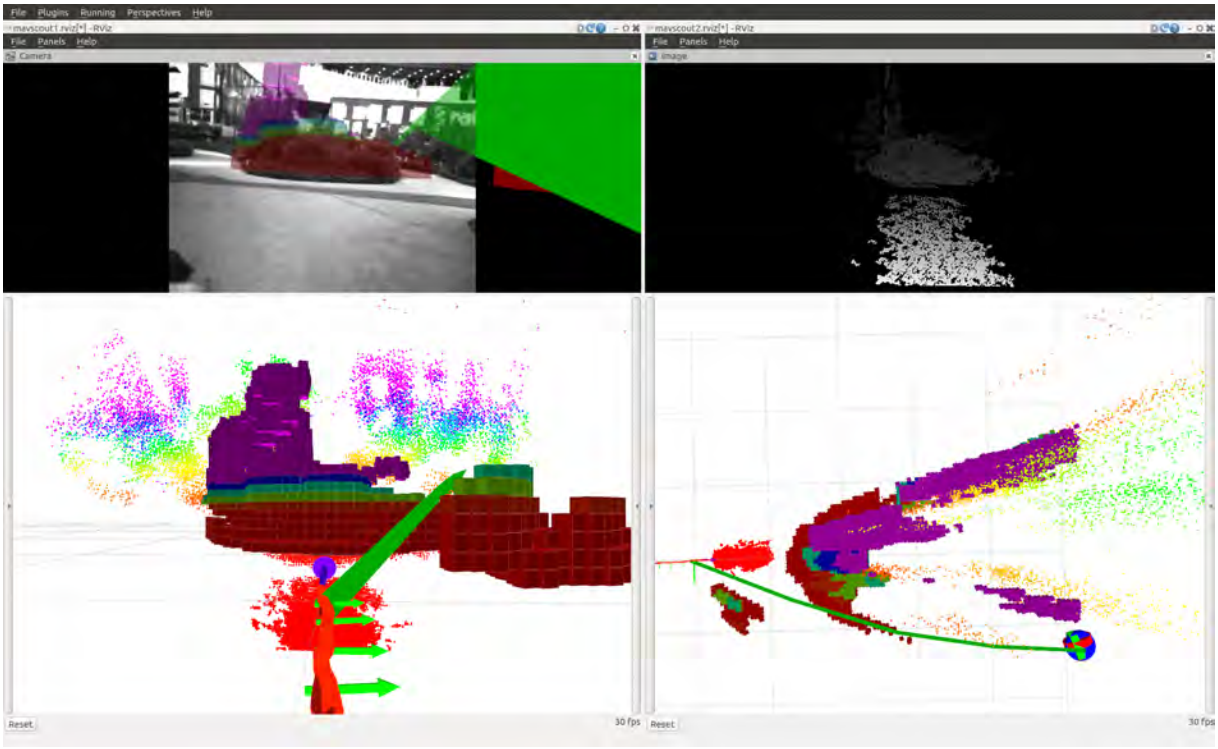


Figure 5.5: Initial plan to right hand side street seen in the camera.

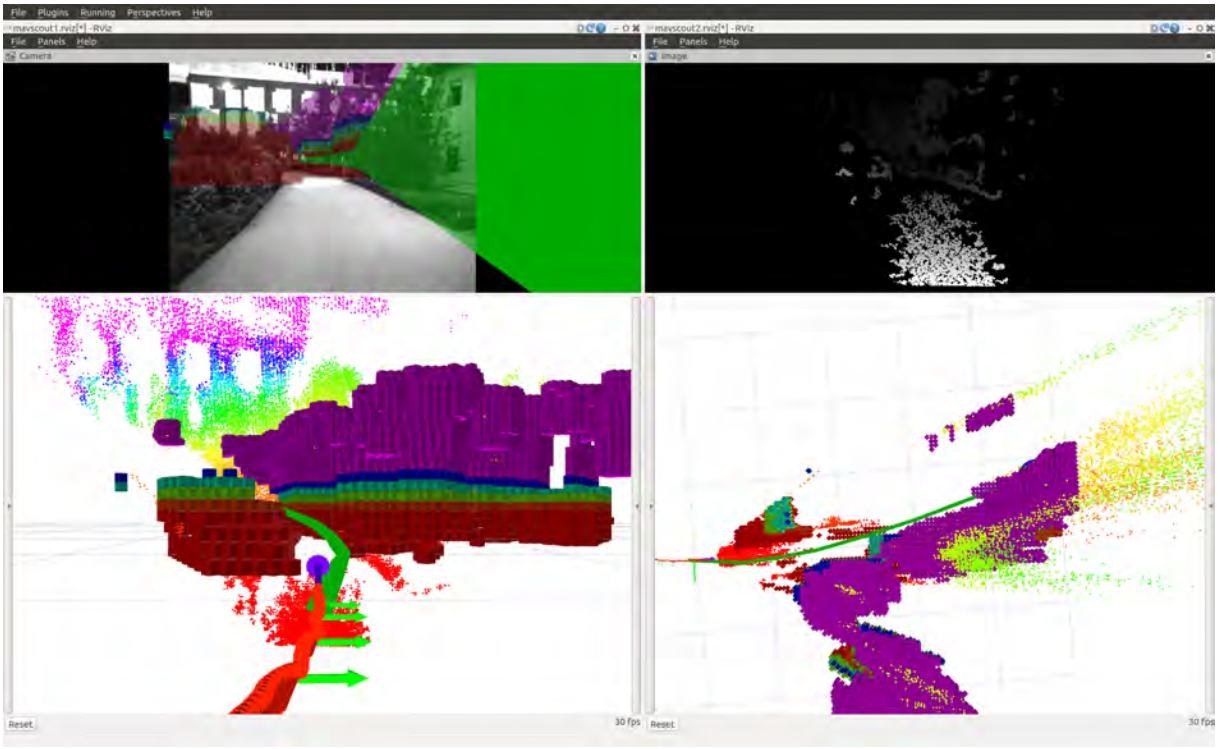


Figure 5.6: Replanned path upon entering the street as more information in fused into the map.

## 5.3 Experiments

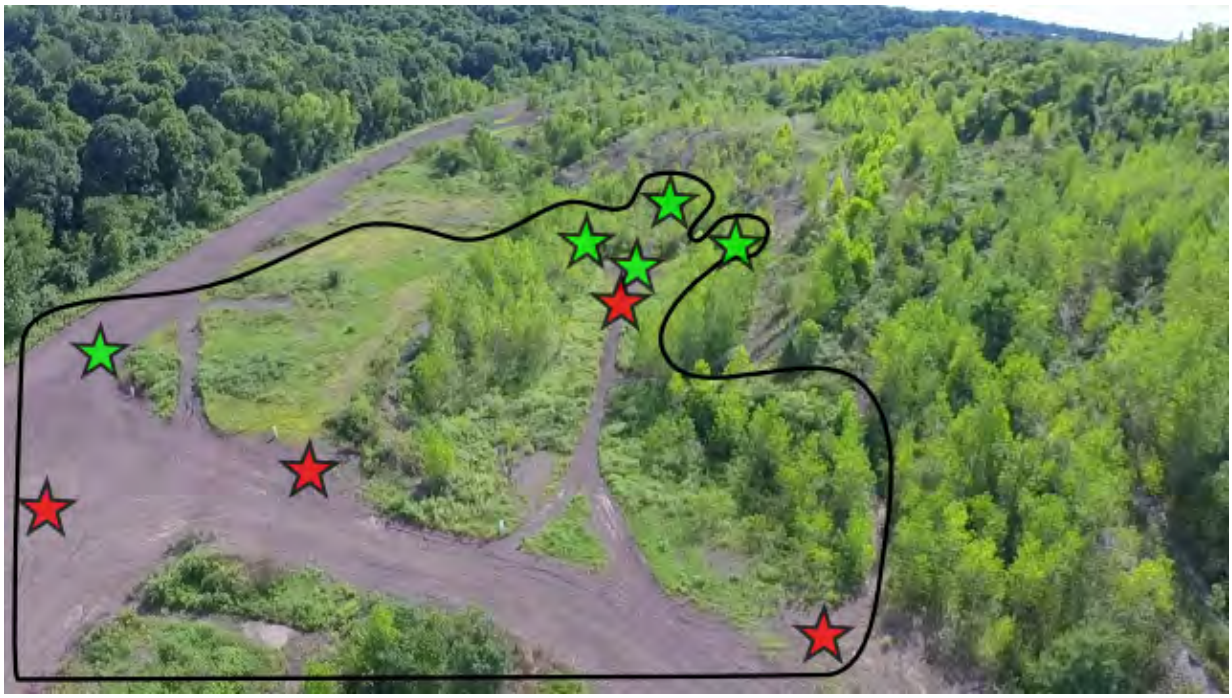


Figure 5.7: Marked area of the location where experiments were carried out. Various start and goal locations are shown in green and red star markers respectively. Runs were conducted between random pairs of the start-goal locations and sometimes by creating a sparse waypoint list using the marked goals.

We conducted most of the experiments in the highlighted area shown in Figure 5.7. Some of the features of region were narrow trails, dense foliage and varying height tree line, all of which made for challenging and interesting obstacles. Tests involved manual take-off and sending a list of sparse global waypoints to the obstacle avoidance system with the desired velocity. Sparse global waypoints allowed obstacle avoidance system to plan around obstacles determining vehicles path and safety. Table(5.1) lists the values we used for conducting the experiments.

Table 5.1: Parameters Used

| Parameter | Value |
|---|---|
| Baseline: $b$ | 0.35m |
| Focal length: $f$ | 514.17 pixels |
| Correspondence error: $\sigma$ | 0.5 |
| Connected component range: $range$ | $2r_v$ |
| Robot radius: $r_v$ | 1.5m |
| Lenient Occupancy Threshold: $\gamma_{high}$ | 1.8 |
| Strict Occupancy Threshold: $\gamma_{low}$ | 0.9 |
| No. of nodes in Pose graph: $N_{graph}$ | 10 |
| Displacement between nodes: $\gamma_d$ | 1.5m |
| Angle between nodes: $\gamma_\psi$ | $30^\circ$ |

## 5.4 Results

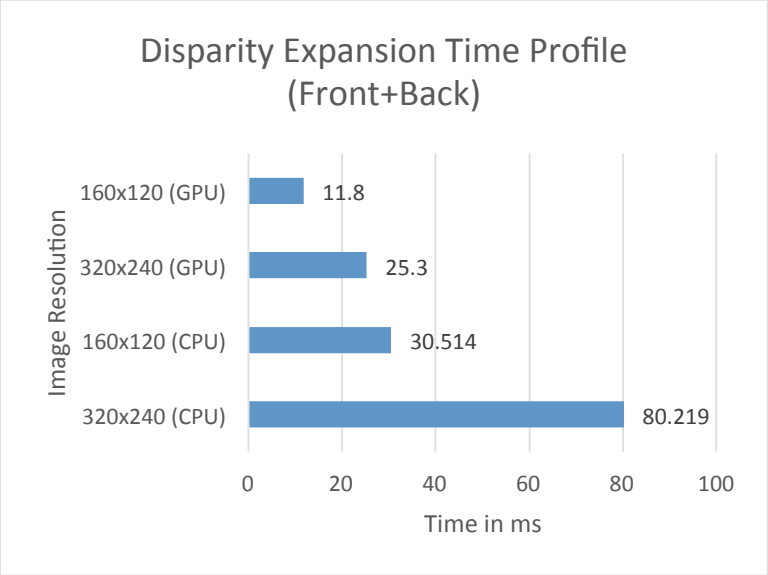



Figure 5.8: Time profile of expansion step on a Jetson TK1 SoM.

Figure 5.8 shows the time taken to process a single disparity image to compute the frontal and back expansions using Algorithm(1) on the onboard ARM computer. In our experiments we used CPU version at $320 \times 240$ resolution because the GPU was used for semantic classification algorithm as concurrent part of the experiments. A pose graph using Algorithm(2) was created and used for collision checks using equation(3.11). Using our approach a single occupancy inference and collision check takes on average only $4\mu s$.

Figure 5.9 and Figure 5.20 show the overall time taken to find a new path. It can be seen for $10m/s$ runs maximum of $198ms$ were used, given there was a path. This is undesirable and will be discussed further in Section 5.4.2.

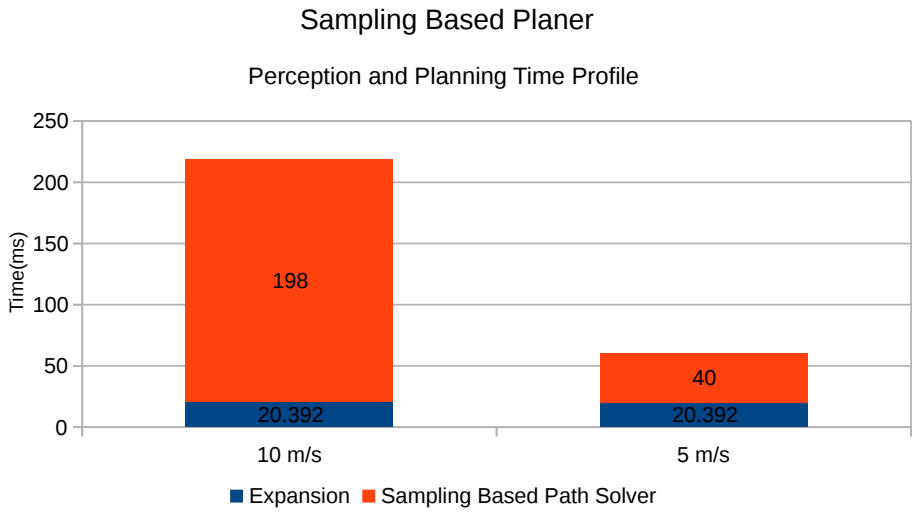



Figure 5.9: Time Profile for sampling based planner approach on Jetson TX2. This plots the maximum time taken given there is a path.

### 5.4.1 $4m/s$ Runs

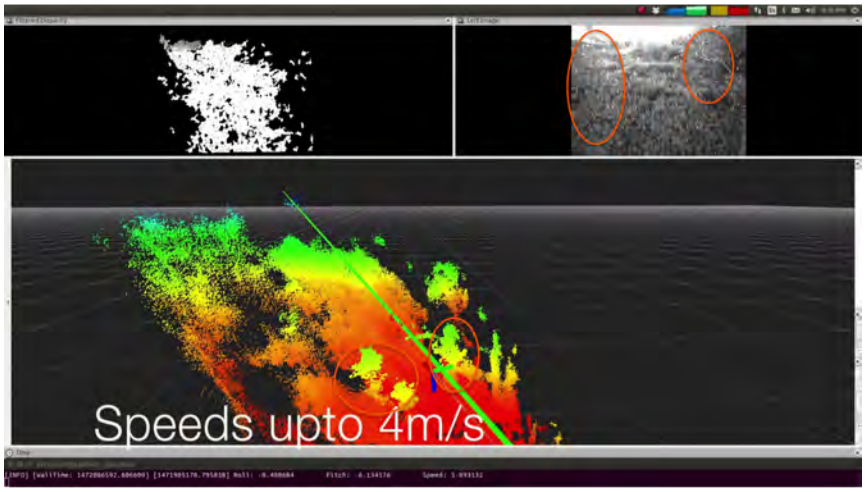
Figure 5.10(a) Shows planned path going through two low height trees. The top left is the disparity image with left camera image shown on top right. The point cloud is only for visualization purpose and the trees are marked in red ellipses. Although the trees are not completely visible in the current disparity image, they are still a part of obstacles as they were seen at previous robot positions and hence stored in the pose graph. Without the pose graph these trees would have been invisible to the robot. Thus the pose graph helps in keeping memory of obstacles which were seen previously but can't be observed as they exceed the limit of maximum possible disparity after robot motion.

Figure 5.10(b) shows the previous path was replanned and pushed up as more observations of the bushes/trees are made at long range are marked as obstacles at approximately $30m$ distance from the robot. This was possible due to fusion of occupancy using several disparity images in the pose graph.

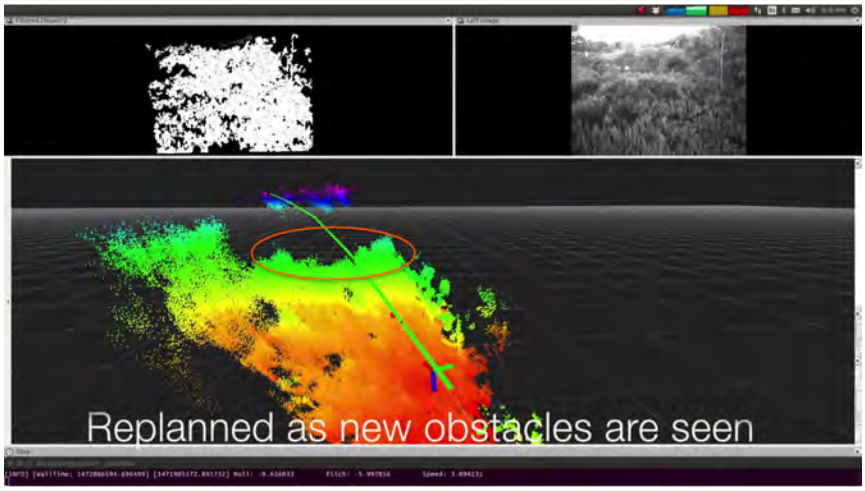
Figure 5.10(c) emphasises the advantage of planning in disparity space at long distances. At greater distances the point cloud is very noisy but we are able to get some information about occupancy by using all the sensor data. While occupancy grids would have huge impact, both memory wise and computationally to use all this data, our approach is able to incorporate all the information using minimalistic image space representation and do better occupancy inference.

Figure 5.11 shows the reactive nature of our approach. For this experiment the robot was allowed to find a plan outside the sensor's field of view and was given a goal point in right direction. As the robot follows the plan and turns right, an obstacle obstructing its path is detected and a new plan avoiding it is generated. This happened at a speed of $4m/s$ hence implying our approach quickly reacts to newly seen obstacles. Figure 1.2 shows the third person view of the same run.

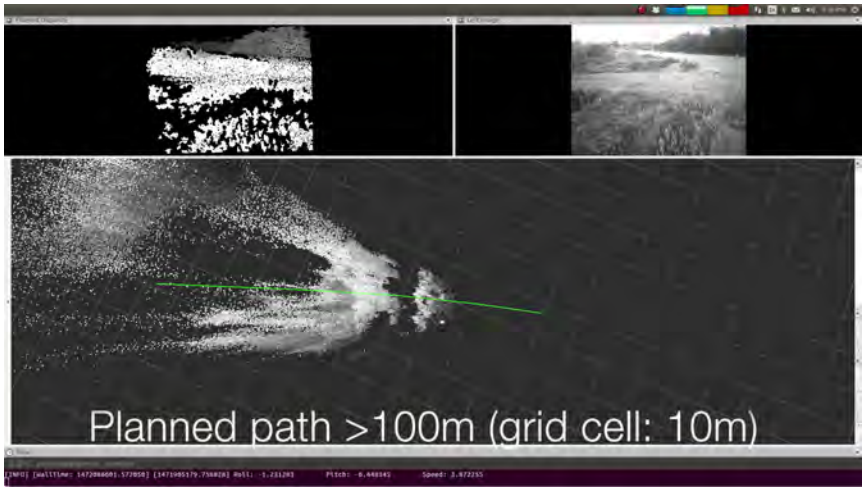
Using the parameters specified in Table 5.1, if the robot moves $15m$ maintaining 10 nodes




(a) Planned path(green) between low trees highlighted in red ellipses.




(b) Replanned(green path) as more observations are made, marked in red ellipse.




(c) Long range planning horizon. The point cloud shows the noisy measurement but even noisy information allows to infer occupancy at long distances.

Figure 5.10: Point cloud is shown at the bottom in all three figures for reference. Point cloud is colored by height in (a) & (b) and by actual intensity in (c).

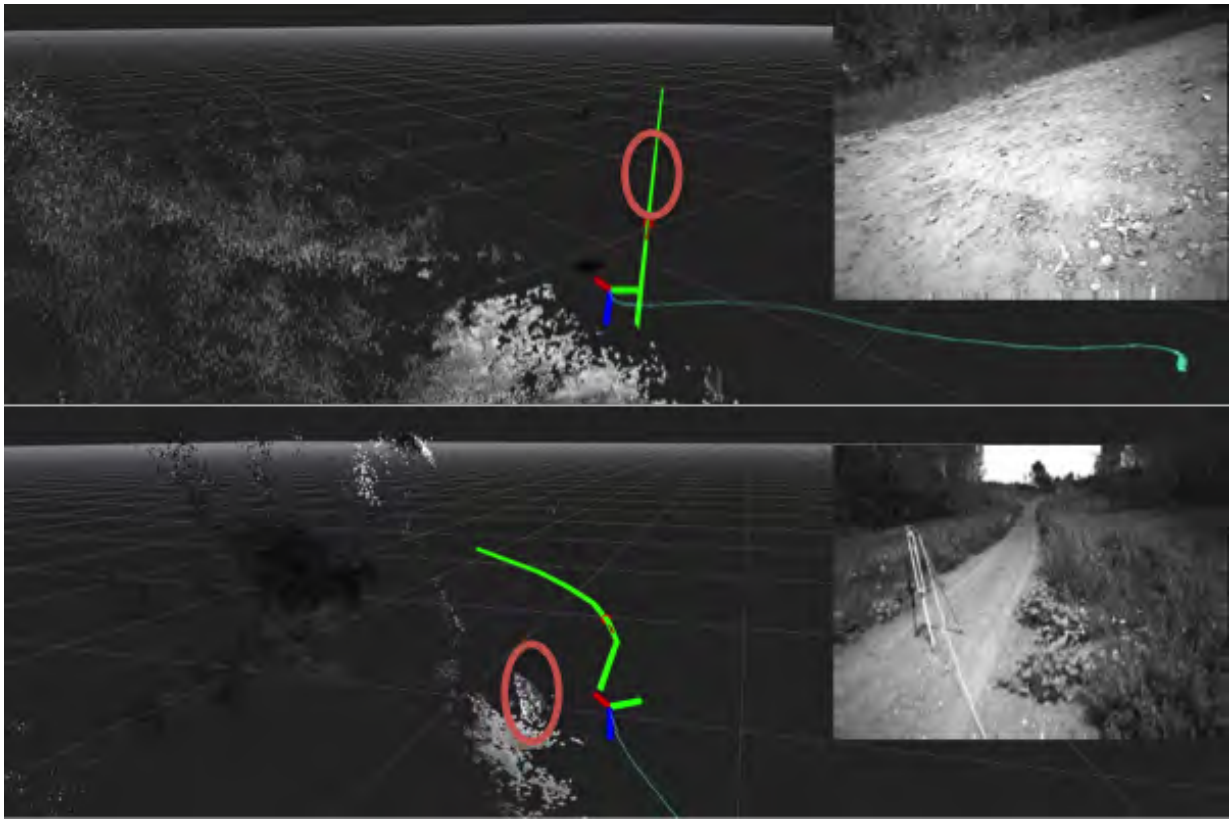


Figure 5.11: Reactive Planning at $4m/s$: Top image shows the robot has planned to go right with unseen obstacle marked in red ellipse. Bottom image: after banking right an obstacle obstructs the previous plan and a new plan avoiding it is generated.

and assuming a maximum of only $100m$ depth ($1.79pixel$ disparity) per image our approach uses approximately $38\%$ of the memory required by a gridmap of cell size $1m^3$ covering the same volume. This is the case when using a gridmap of large cell size meaning a very coarse resolution. For a better resolution gridmap will require even more memory.

More than 100 runs were executed with approximately $1.6$ hours in autonomous mode, covering a cumulative distance of approximately $1.5Km$. The maximum speed was capped at $4m/s$. Our approach allowed us to plan to distances greater than $100m$ as shown in Figure 5.10(c). Average distance to goal was $36m$. The standard deviation of length of planned paths from straight line paths was on average of $1.38m$ with a maximum of $30m$. This shows that in most cases planned paths were close to a straight path but with slight deviation to avoid obstacles.

### 5.4.2 $10m/s$ Runs


Figure 5.12: Quadrotor platform used for $10m/s$ runs: equipped with stereo camera sensor suite and only one onboard ARM computer. This helped in significant weight reduction of about $33\%$, from $1.2Kg$ to $0.8Kg$.

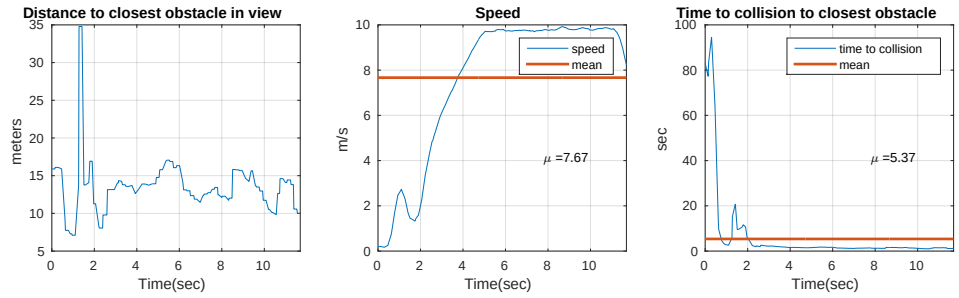



Figure 5.13: $10m/s$ run 1 showing the distance to closest obstacle in current view, current speed and current time to collision to the closest obstacle.

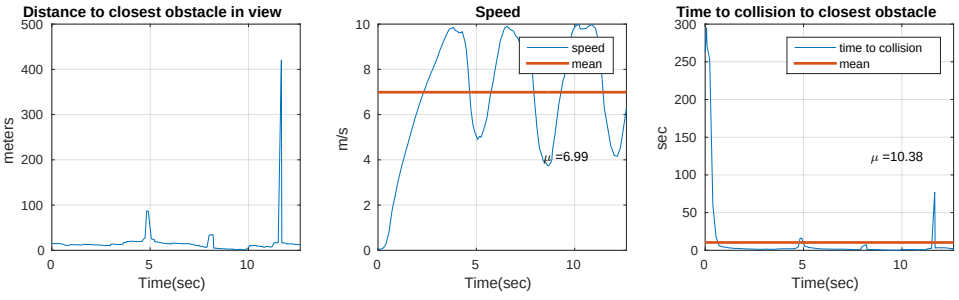



Figure 5.14: $10m/s$ run 2 showing the distance to closest obstacle in current view, current speed and current time to collision to the closest obstacle.

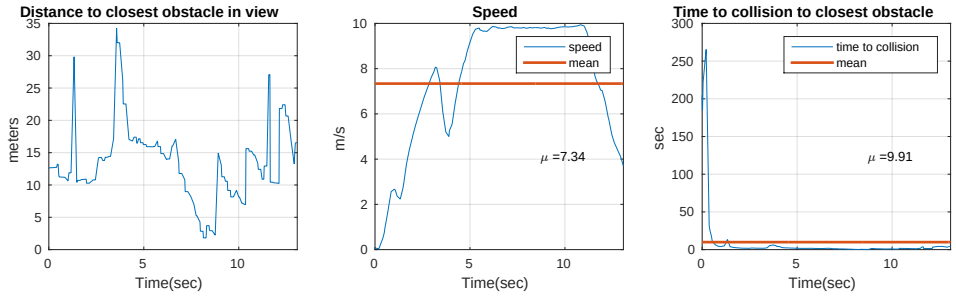



Figure 5.15: $10m/s$ run 4 showing the distance to closest obstacle in current view, current speed and current time to collision to the closest obstacle.

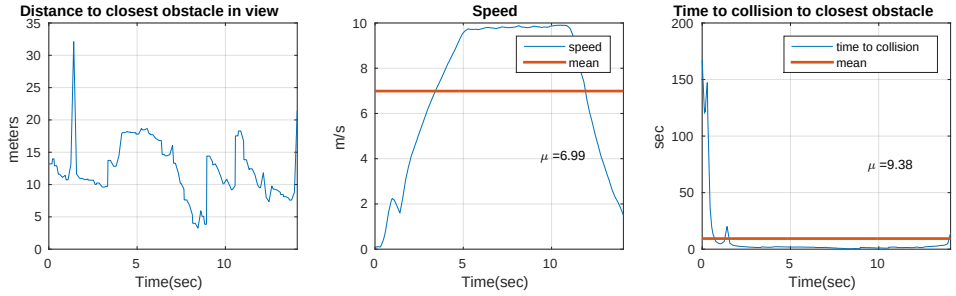



Figure 5.16: $10m/s$ run 5 showing the distance to closest obstacle in current view, current speed and current time to collision to the closest obstacle.

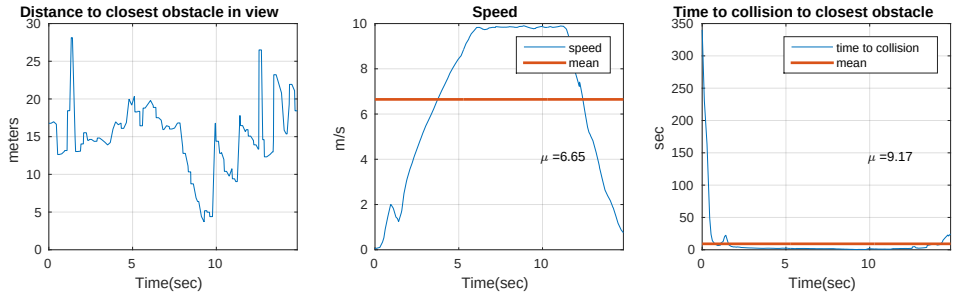



Figure 5.17: $10m/s$ run 6 showing the distance to closest obstacle in current view, current speed and current time to collision to the closest obstacle.

Figure 5.13 to Figure 5.17 show plots for the distance to closest obstacle in the current field of view of the sensor during the run. The time to collision plots show the time to collide with the closest obstacle if the robot were to fly in the direction of the obstacle. It can be seen that the time to collision is high in beginning and then falls because the robot flies to longer distances and

mostly looking down at nearby obstacles not necessarily on the flight path. The speed plots show that the robot flies at the maximum set speed of $10m/s$. Some speed plots do not end at zero velocity because when the robot almost reached the goal which was usually located far from the flight operator and hence had to take manual control. The plots only show data for autonomous modes only.

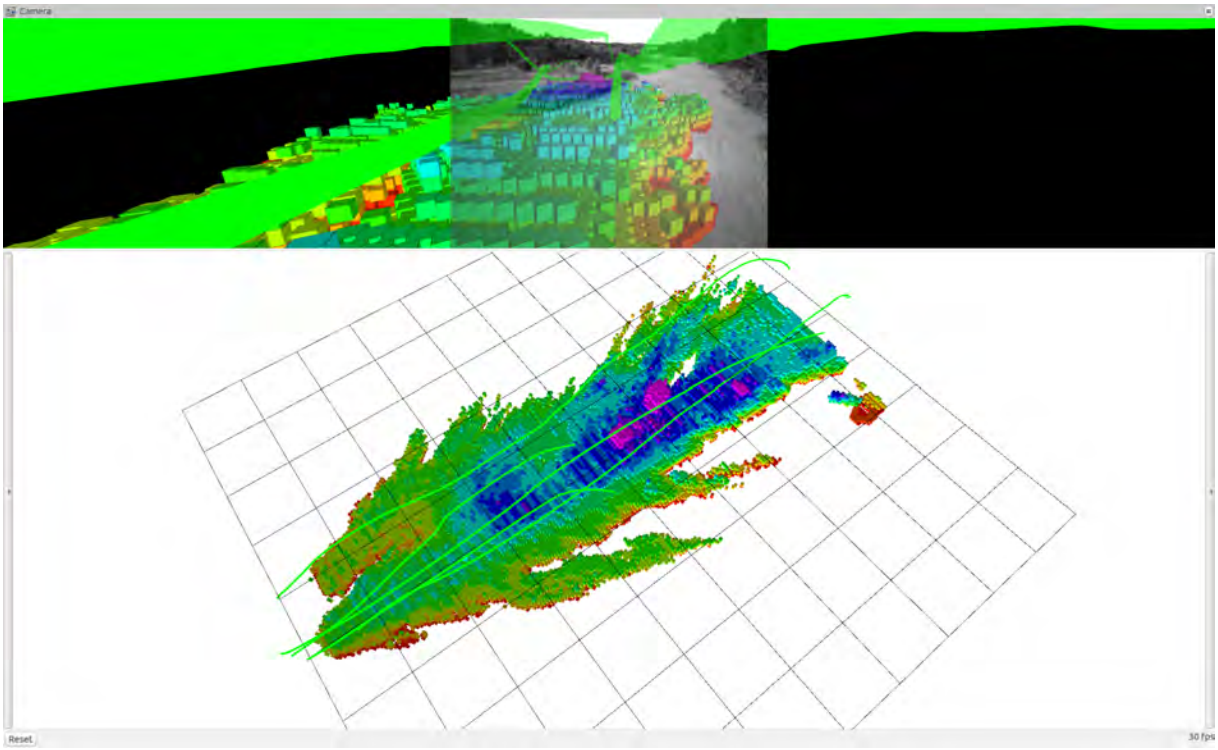


Figure 5.18: $10m/s$ Obstacle avoidance. Top shows the camera view with projected tracked paths and the occupancy map. Occupancy is colored by height. The Occupancy map is the overall result and was not available to the robot during runs.

Figure 5.18 Shows a different set of five runs made at $10m/s$ with first two plans being taken over by the safety pilot as the robot avoided the obstacles too aggressively. The next three runs were manually overridden upon reaching the goal. The goal was given $100m$ ahead of current robot position. We ran into a few issues with these runs as discussed below.

The vehicle takes long time to reach a speed of $10m/s$ and is constantly pitched down at an angle more than $20°$ for over $4sec$ as shown in Figure 4.6. Also for scouting application we already have the sensor boom looking down at an angle of $15°$. Due to this for a long period of time as the vehicle is accelerating, hardly any new information is observed when flying straight ahead. The current horizontal field of view is $64°$. Under such scenario we are almost looking $3°$ below the horizon and hence the robot either perceives new information only upon getting too close to obstacles or if happens to pitch up. Figure 5.18 shows one short run where it successfully avoided the obstacle almost $10m$ before. This is also partially responsible for higher plan times as shown in Figure 5.9 as the obstacle gets too close and it takes longer to find a path in close proximity to the robot.

The other problem we faced was that the vehicle is still heavy to aggressively follow the planned path. In the same short run as discussed above, the generated path avoids the obstacle much early but the robot due its inertia while flying at $10m/s$ gets closer to obstacle before steering away.

### 5.4.3 Trajectory Library based planner runs


Figure 5.19: A smaller quadrotor platform used for experiments: equipped with stereo camera sensor suite and onboard ARM computer

We test our algorithms on-board an autonomous UAV system, see Figure 5.19. The base platform is an off-the-shelf quadrotor vehicle retrofitted with in-house developed sensing and computing suite designed consisting of a stereo camera pair an integrated GPS/INS unit. The stereo camera pair provides $640 \times 512$ resolution image which is used to compute a disparity image at $10$ fps for the obstacle avoidance. All computation for autonomous operation is performed onboard.

Table 5.2: Parameters Used

| Parameter | Value |
|---|---|
| Baseline: $b$ | 0.17m |
| Focal length: $f$ | 514.17 pixels |
| Correspondence error: $\sigma$ | 0.5 |
| Connected component range: $range$ | $2r_v$ |
| Robot radius: $r_v$ | 0.5m |
| Lenient Occupancy Threshold: $\gamma_{high}$ | 1.8 |
| Strict Occupancy Threshold: $\gamma_{low}$ | 0.9 |
| No. of nodes in Pose graph: $N_{graph}$ | 10 |
| Displacement between nodes: $\gamma_d$ | 1.0m |
| Angle between nodes: $\gamma_\psi$ | $30^\circ$ |

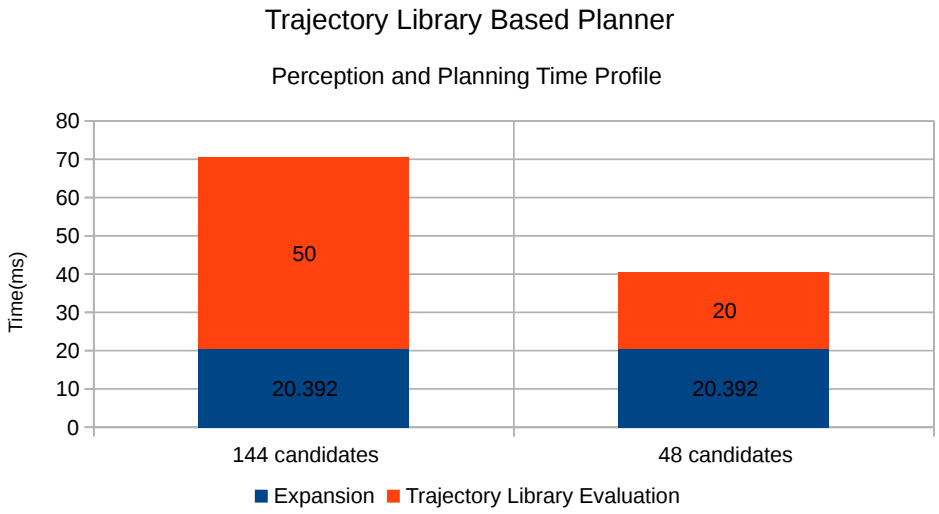



Figure 5.20: Time Profile for trajectory library based planner approach on Jetson TX2.

We conducted indoor and outdoor experiments using a set of trajectory library. Tests involved autonomous take off, navigate to pre-fixed well separated global waypoints and finally land. Table(5.2) lists the values we used for conducting the experiments. Figure 5.20 shows the affect on compute times on varying the size of candidates in the trajectory library.

Most runs were conducted between the speed of $2m/s$ to $3m/s$. During the Robotics Week, 2017 at RI-CMU more than 100 runs were conducted in front of public with no failures. The speed was restricted to $2.5m/s$.

Figure 5.21 shows the setup. The Start and Goal are separated by $10m$ with a curvy constructed obstacle path.

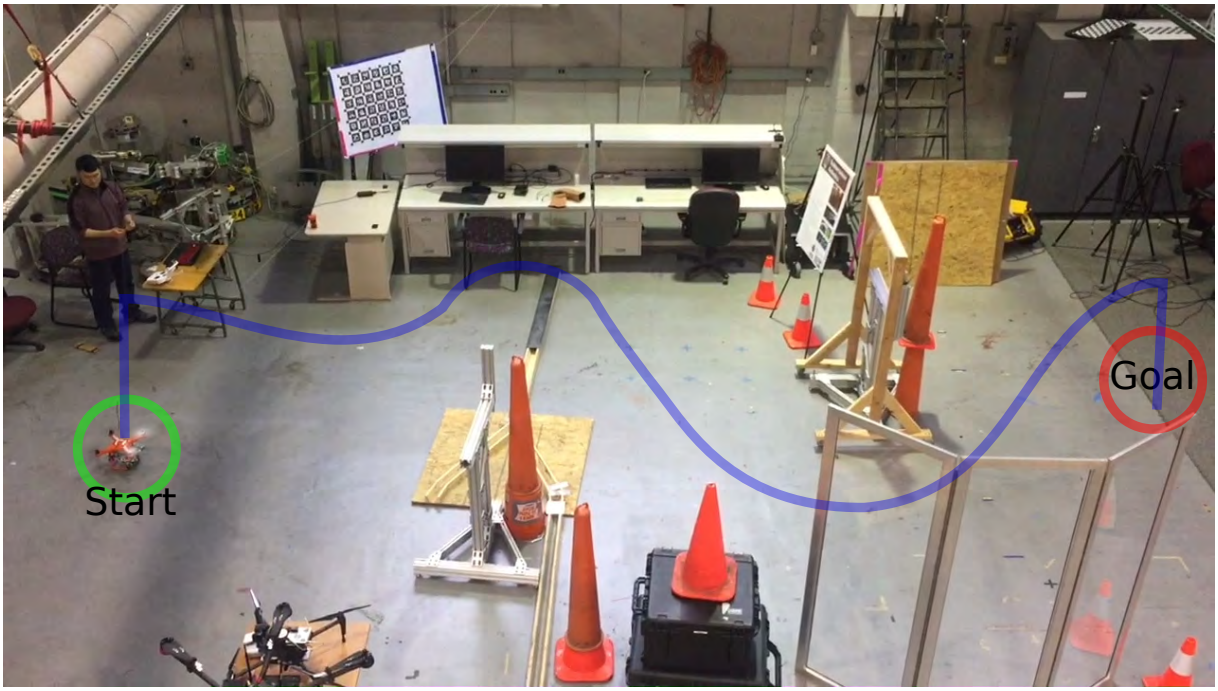



Figure 5.21: Demo setup for Robotics Week, 2017 at RI-CMU. The Start is on left and Goal is $10m$ away on right with several obstacles restricting a direct path, hence forming a curvy corridor to follow. We did more than 100 runs with no failures at $2.5m/s$

## 5.5 Limitations

In this section we discuss some limitations of the disparity based obstacle representation.

1. **Overriding:** Currently we only expand the obstacles seen closer and thereby obstacles that are already seen will not be correctly expanded. Figure 5.22 illustrates this issue. The problem is the obstacle is seen in the original disparity image but since only a frontal expansion and a connected component back expansion disparity image are maintained in our representation, we have to drop the already seen obstacle which is not connected to the obstacle seen closer to the sensor. We can address this issue by atleast using four layers of expanded images to completely capture the information available in one disparity image frame.
2. **Expansion Outside FOV:** This is caused due to fixed size expanded disparity image also illustrated in Figure 5.22. This causes obstacles seen at the border of the current image not getting expanded outside the image size. Unlike the previous issue where some information is dropped, this issue is more due to the fixed size image data-structure. One way to address this issue can be to dynamically change the size i.e. width and height of the expanded disparity images to completely expand obstacles seen on edges. However, we will need to be careful about the camera intrinsics for the new adaptive image size.

The following Figure 5.22 illustrates this.

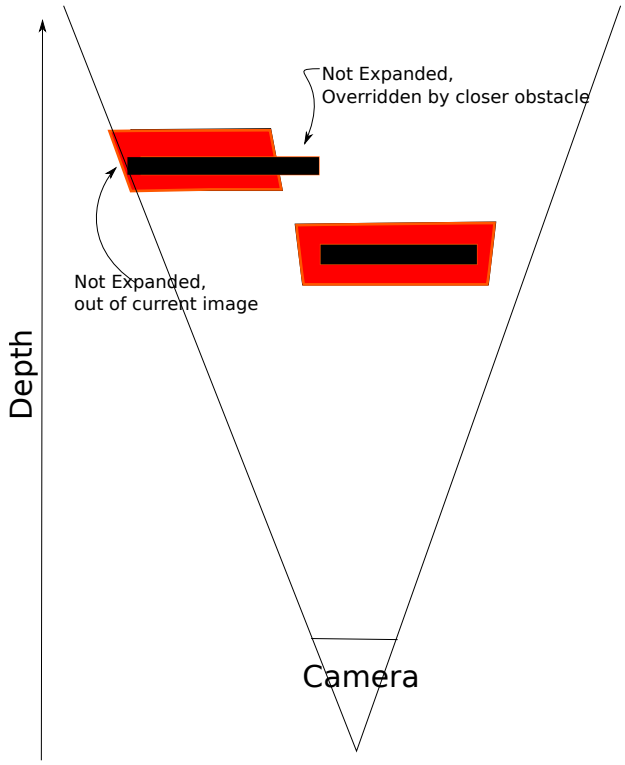



Figure 5.22: Black is obstacle and red is expansion. The two limitations of closer obstacle expansion overriding the backward obstacle expansion and expansion restriction to within FOV are shown.

# Chapter 6

# Conclusion & Future Work

## 6.1 Conclusions

We have presented an approach and a system design that allows high speed, non-myopic obstacle avoidance. We demonstrated the system flying at $10m/s$ in dense foliage while relying on stereo image data for modeling the world. To our knowledge it is the first one to do so.

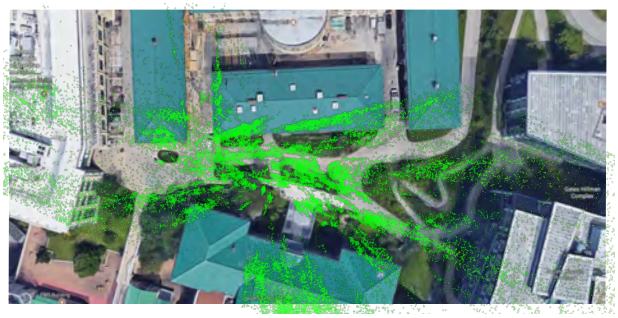


Figure 6.1: Shows an octomap built using full stereo point cloud projected on google-map of RI-CMU.

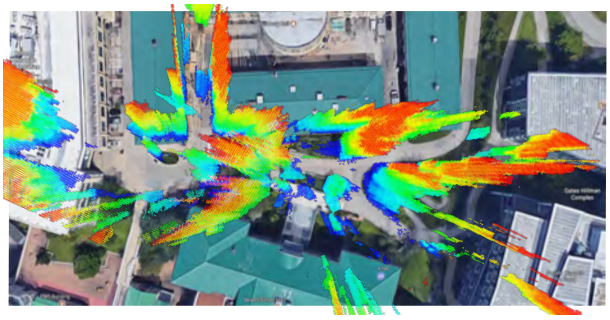


Figure 6.2: Shows a voxel map generated using our approach projected on google-map of RI-CMU.

The key factor that enables our system to perform safely at high speeds in highly cluttered environments is integrating multiple stereo sensor frames in real time while reasoning about the

related highly non-linear noise model to generate a world representation in inverse depth space. The current state-of-the-art methods e.g OctoMaps fail to properly incorporate complete stereo camera data as apparent from Figure 6.1 while our method performs much better as shown in Figure 6.2 in context of obstacle mapping.

Using FPGA hardware for disparity calculation, combined with fast disparity expansion allowed us to limit our computational burden. This allowed the system to share computation with classification and state estimation tasks.

We believe that our approach is suitable for small MAVs with limited payload carrying capabilities to enable good obstacle avoidance performance.

## 6.2 Future Work

The current disparity space representation lacks an explicit model of unknown space, rendering the system vulnerable to collision whilst operating in environments with complex geometries. We are currently working on this issue by performing disparity space expansion over multiple layers by extending the disparity expansion step from two: front and back surface layers to four, to account for the limitation discussed in Figure 5.22. Figure 6.3 illustrates this.

Currently our occupancy inference uses a confidence based metric instead of computing the probability mass for faster computation. We should be able to get better results by using the probability mass by approximating it using a piecewise linear interpolant.

The current pose graph only maintains spatial memory and we would extend it to have some temporal decay to forget non-static obstacles that happened to be a part of a keyframe. The other extension could be to improve the previous keyframes using current observations until next keyframe is captured.

To guarantee vehicle safety we will employ emergency maneuver libraries [3] in conjunction with active control of heading along the lines of the sensor planning approach suggested in [2].

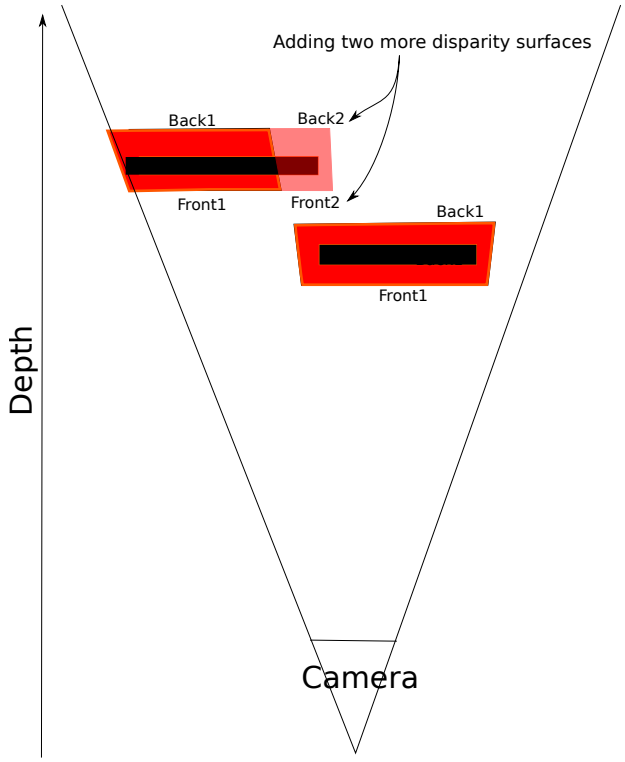



Figure 6.3: We can use four channel image instead of current two channels to address the limitation shown in Figure 5.22

# Bibliography


[1] Franz Andert and Florian Adolf. Online world modeling and path planning for an unmanned helicopter. *Autonomous Robots*, 27(3):147–164, 2009. 2.1

[2] Sankalp Arora and Sebastian Scherer. Pasp: Policy based approach for sensor planning. In *IEEE, International Conference on Robotics and Automatio*, 2015. 6.2

[3] Sankalp Arora, Sanjiban Chowdhury, Daniel Althoff, and Sebastian Scherer. Emergency maneuver library - ensuring safe navigation in partially known environments. In *IEEE, International Conference on Robotics and Automation*, 2015. 6.2

[4] A. J. Barry and R. Tedrake. Pushbroom stereo for high-speed navigation in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3046–3052, May 2015. doi: 10.1109/ICRA.2015.7139617. 1.1, 2.1

[5] Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Grasp planning in complex scenes. 4.3

[6] D Dey, TY Liu, B Sofman, and JA Bagnell. Efficient Optimization of Control Libraries. *Aaai*, pages 1983–1989, 2012. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/download/5081/5361`. 4.3

[7] Debadeepta Dey, Kumar Shaurya Shankar, Sam Zeng, Rupesh Mehta, M Talha Agcayazi, Christopher Eriksen, Shreyansh Daftry, Martial Hebert, and J Andrew Bagnell. Vision and learning for deliberative monocular cluttered flight. In *Field and Service Robotics*, pages 391–409. Springer, 2016. 1.1

[8] E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)*, volume 1, pages 821–826 vol.1, 2000. doi: 10.1109/CDC.2000.912871. 4.3

[9] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. *CoRR*, abs/1405.5848, 2014. URL `http://arxiv.org/abs/1405.5848`. 4.2

[10] Pascal Gohl, Dominik Honegger, Sammy Omari, Markus Achtelik, Marc Pollefeys, and Roland Siegwart. Omnidirectional visual obstacle detection using embedded FPGA. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem:3938–3943, 2015. ISSN 21530866. doi: 10.1109/IROS.2015.7353931. 2.1, 3.1

[11] Corey Goldfeder, Matei Ciocarlie, Jaime Peretzman, Hao Dang, and Peter K. Allen. Data-driven grasping with partial sensor data. 4.3

[12] Colin J. Green and Alonzo Kelly. Toward optimal sampling in the space of paths. *Springer Tracts in Advanced Robotics*, 66(STAR):281–292, 2010. ISSN 16107438. doi: 10.1007/978-3-642-14743-2_24. 4.3

[13] Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 2472–2477. IEEE, 2011. 2.1

[14] Gabriel M Hoffmann, Steven L Waslander, and Claire J Tomlin. Quadrotor helicopter trajectory tracking control. In *AIAA guidance, navigation and control conference and exhibit*, pages 1–14, 2008. (document), 4.4, 4.7

[15] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013. ISSN 1573-7527. doi: 10.1007/s10514-012-9321-0. URL `http://dx.doi.org/10.1007/s10514-012-9321-0`. 2.1

[16] 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)*, pages 3242–3249, May 2014. doi: 10.1109/ICRA.2014.6907325. 1.1, 1.2, 2.1, 3.1, 3.1.5, 4.1

[17] Konstantin Schauwecker. Sp1: Stereo vision in real time. 3.1.2, 5.1

[18] Sebastian Scherer, Joern Rehder, Supreeth Achar, Hugh Cover, Andrew Chambers, Stephen Nuske, and Sanjiv Singh. River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, 33(1-2):189–214, 2012. 1.1

[19] Martin Stolle and Chris Atkeson. Policies based on trajectory libraries. In *In Proceedings of the International Conference on Robotics and Automation (ICRA*, 2006. 4.3

[20] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623. 3.1.7