PLANNING AND CONTROL ALGORITHMS FOR AUTONOMOUS FIELD ROBOTS

A Project Report

submitted by

S.R.MANIKANDASRIRAM

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
&
MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **PLANNING AND CONTROL ALGORITHMS FOR AUTONOMOUS FIELD ROBOTS**, submitted by **S.R.MANIKANDASRIRAM**, to the Indian Institute of Technology, Madras, for the award of the degree of **BACHELOR OF TECHNOLOGY & MASTER OF TECHNOLOGY**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Bharath Bhikkaji
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Prof. Bharath Bhikkaji for his timely guidance and freedom to explore and learn different aspects of robotics throughout the last two years. Over and above the technical guidance, I thank him for his positive and encouraging outlook towards carrying out research. I would also like to thank Prof. Jerome Le Ny (Polytechnique Montreal) for guiding me through the later part of this project and giving me an opportunity to work in CIPHER lab during the summer of 2015 under the Mitacs Globalink program. In this regard, I am also thankful to Mitacs Canada for funding my summer research internship. I wish to also thank Raghav H.V and Saad Chidami for their help and support at various stages of the project. Finally, I thank my family, friends and the institute for providing me with support, opportunity and resources to carry out this research work.

ABSTRACT

KEYWORDS: Testbed; motion planning; differential games; target guarding; autonomous mapping; active SLAM

This thesis is composed of two parts where we have addressed two different problems in the broad domain of autonomous field robots. The first problem deals with evaluating planning and control algorithms for an autonomous guard robot which must protect a bounded target region from intruders. Most analytic solutions for this problem assume a 2D holonomic robot while most robots are in general non-holonomic. In this work, we study the effects of vehicle dynamics on the performance of these analytic solutions. The second problem deals with developing motion planning strategies for autonomously mapping a structure of interest using a ground robot equipped with a depth sensor. We develop algorithms that do not assume any prior information about the structure including size or geometry of the structure. Moreover, our proposed policies have been verified using simulations and were found to achieve a higher coverage than some classic strategies like frontier based exploration algorithm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

3D	Three Dimensions
CAD	Computer Aided Design
CE	Cavity Exploration
DOF	Degree of Freedom
FBE	Frontier based Exploration
FoR	Frame of Reference
FOV	Field of View
GPS	Global Positioning System
HSV	Hue Saturation Value
ICP	Iterative Closest Point
IITM	Indian Institute of Technology Madras
IMU	Inertial Measurement Unit
LIDAR	LIght Detection And Ranging
PCA	Principal Component Analysis
PCL	Point Cloud Library
PE	Perimeter Exploration
PID	Proportional Integral Derivative
RGB	Red Green Blue
ROS	Robot Operating System
RTAB-Map	Real-Time Appearance Based Mapping
SfM	Structure from Motion
vSLAM	visual Simultaneous Localization and Mapping

Part I

Autonomous Guarding of a Bounded Region

CHAPTER 1

Introduction

An autonomous guard robot can be used in a variety of fields ranging from anti-poaching systems, where autonomous robots protect the habitat of wild animals from intruding poachers, to border patrol problems, where teams of autonomous robots are tasked with protecting the border of a secluded campus or region of interest. A major challenge in building such a guard robot is developing path planning algorithms that can compute optimal trajectories for the robot in real-time. Rufus Isaacs, in his pioneering work on differential games (Isaacs, 1999), presented optimal strategies for both the guard and the intruder, assuming both agents to have equal speed, see Fig. 1.1.

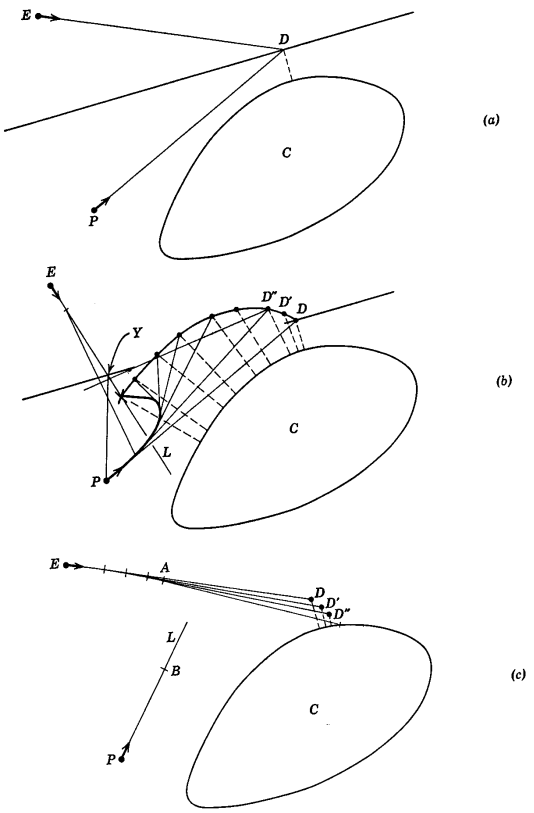


Figure 1.1: Guarding a target problem from (Isaacs, 1999). a) P and E plays optimally. b) P plays optimally, but E does not. c) E plays optimally, but P does not.

The target guarding problem is a two player game wherein the intruder robot (evader E) attempts to reach a secluded region (target area C) while evading capture by the guard robot (pursuer P). The objective of the participating agents are opposite, thus making it

a differential game. While the intruder attempts to minimize the distance to the target area at the end of the game, the guard robot attempts to maximize it. The game ends if the intruder safely reaches the target area or if the guard robot successfully intercepts the intruder before the intruder reaches the target area. Rufus Isaacs showed that, for the simple case where both agents have equal speeds, if the target region is closer to the guard, then the optimal strategy for both agents is to move towards that point D, on the perpendicular bisector of their initial positions, which is closest to that target area. In particular, this implies that if the intruder follows a pure-pursuit strategy (i.e.) head directly towards the target, he is playing sub-optimally. A key characteristic of a differential game is that if either of the agents adopt a policy other than the optimal strategy, it results in an advantage for the other agent. Therefore, an optimal guidance law for the guard robot should compute the desired heading for the robot at each instant using the current positions of the agents.

In the simple version of the game analysed by Rufus Isaacs, the velocities of both the agents were assumed to be equal. Further, the criterion for capture is the coincidence of the two robots. In reality, both these assumptions are incorrect. The velocity of the guard robot is usually greater than that of the intruder robot. Further, the robots are not point objects. Therefore, their physical size implies that capture occurs if the agents come within a certain distance δ whose value depends on the size of both the agents. In (Venkatesan and Sinha, 2014), Raghav et. al. extended the work of Rufus Isaacs to remove these two assumptions. Further, they provide a fast computation method to compute the optimal heading for the guard robot at every instant. In (Lau and Liu, 2014), Liu and Lau present an autonomous border patrol system that considers a single fast guard robot protecting the target region from multiple non-cooperating intruders. A major limitation of these works is that they assume the agents to be point objects that can instantaneously move in any direction on the 2D plane. But such robots belong to a special class of robots called holonomic robots. Most robots in reality are non-holonomic and have a minimum turning radius. Even robots with zero minimum turning radius, such as differential drive robots, have constrained motion such as inability to move laterally. Such constraints on the dynamics of the robots can potentially change the outcome of the game. In this work, we validate the optimal strategies for the guard robot proposed in (Venkatesan and Sinha, 2014) and study the effect of the vehicle dynamics on the optimality of the proposed strategies.

As a pre-requisite to carrying out this experimental study, we have developed a generic testbed for studying motion planning algorithms. The testbed consists of three differential drive robots constructed using LEGO Mindstorms EV3 kits. The versatility offered by these kits allow for building different configurations for the robot such as three wheeled omni-directional robots, skid-steering based car-like robots among others. We have developed software specific to the differential drive configuration but the same can be easily extended to other drive configurations as well. The testbed is developed following the Robot Operating System (ROS) framework thereby transforming the Mindstorms kits into a low-cost research platform. The LEGO robots are controlled using the standard protocols followed by ROS and therefore, all the algorithms (packages) available in ROS can be used in conjunction with the testbed. In addition to developing drivers that interface LEGO robots with ROS, we have also developed an indoor localization system using a ceiling mounted RGB camera which can detect specially designed coloured markers placed on top of the robots. The testbed has been developed independently and can be used to study a variety of path planning algorithms.

1.1 Contribution of the thesis

In summary, the key contributions of this part of the thesis are:

- a generic testbed for studying path planning algorithms using multiple LEGO EV3 differential drive robots and an indoor localization system using a ceiling mounted camera and specially designed coloured markers
- an experimental validation of the optimal strategies for an autonomous guard robot (target guarding problem)
- a study on the effect of vehicle dynamics on the performance of these analytic solutions

1.2 Chapter wise Descriptions

The rest of this part of the thesis is organized as follows. In Chapter 2, following a brief introduction in Section 2.1, Section 2.2 explains the low-level motion controllers present in EV3 robots. In Section 2.3, we present our vision based localization algorithm for determining the position and orientation of the robots. Next, in Section 2.4,

we explain how the robots can be controlled using the ROS framework. Finally, we discuss some avenues for future work in Section 2.5. In Chapter 3, Section 3.1 provides a brief overview of the problem formulation and the solution provided by Raghav et. al. In Section 3.2, we present a new localization method for the robots to improve the performance of the testbed. The experimental results that show the performance of the policies under the motion constraints of a real robot are presented in Section 3.3. Finally, we conclude in Section 3.4 along with a brief on avenues for future work.

CHAPTER 2

A Generic Testbed for Planning Algorithms

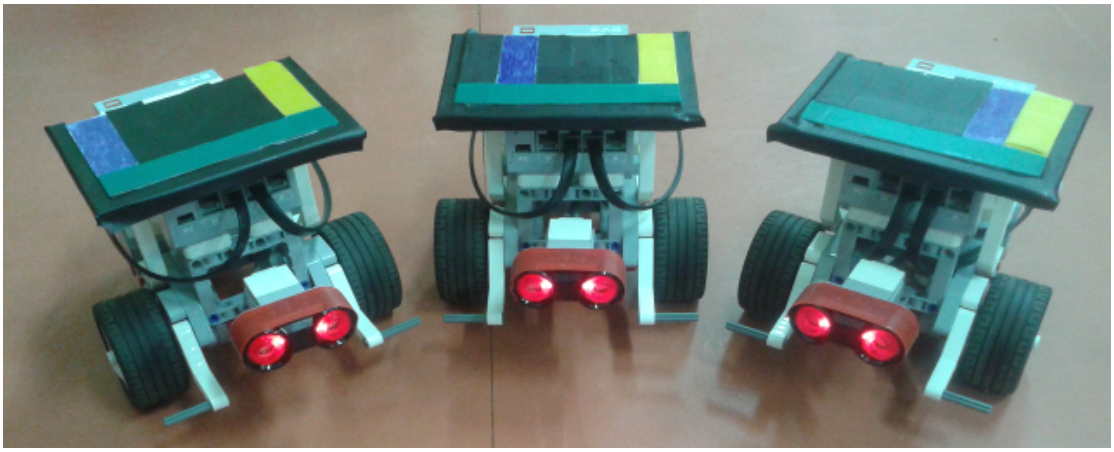


Figure 2.1: Three differential drive robots made using LEGO Mindstorms EV3 kits with coloured markers for uniquely identifying the individual robots.

2.1 Introduction

The objective of this work is to design and build a testing platform using LEGO Mindstorms kits that would help in validating algorithms in the fields of Artificial Intelligence, Control systems and other motion planning techniques. The platform consists of three LEGO robots and a ceiling mounted camera with a birds-eye view. Each LEGO robot has a differential drive system and an ultrasonic range finder for obstacle detection. This system can be easily extended to more number of robots. The localization module and low-level motion controllers are made accessible to the user. The localization is performed using the birds-eye view camera and the low-level motion controllers ensure that the robot tracks a prescribed reference signal. These reference signals are typically generated using planning algorithms that need to be validated.

The entire source-code for the system is publicly available via GitHub through two repositories

1. ev3-ros
   This repository contains the client software which would run in each of the LEGO robots.
2. LEGO Testing platform
   This repository contains the master/server software which would run in the central PC.

The system consists of three modules which would be explained in the subsequent sections

1. Low-level motion controllers
2. Vision based localization
3. Integration with Robot Operating System(ROS) framework

2.2 Low-level motion controllers

The low-level motion controllers are directly implemented on the LEGO EV3 hardware. For this project, a debian based OS, called ev3dev, developed specifically for EV3 has been used. The client software takes velocity commands from the master and calculates the required angular velocities of the individual motors. Then, a separate PID controller for each motor tracks the desired angular velocities.

2.2.1 PID Controller

Each of the LEGO EV3 motor comes with in-built quadrature encoders which measure the rotation of the motors. Using this information as feedback, a standard PID controller is implemented. For this project, the default K_p, K_i, K_d values provided by ev3dev softwere were found to be sufficient. But the ev3dev software provides enough freedom to tune these constants or design a different type of controller as well.

The desired angular velocities for the left and right wheels are calculated from the overall robot velocities using the following transformations

$$v_l = \frac{(v_x + \frac{L}{2} * \omega_t)}{R * \frac{\pi}{180}} \tag{2.1}$$

$$v_r = \frac{(v_x - \frac{L}{2} * \omega_t)}{R * \frac{\pi}{180}} \tag{2.2}$$

where,

v_l : angular velocity of left wheel in deg/s

v_r : angular velocity of right wheel in deg/s

v_x : required linear velocity of robot in m/s

w_t : required angular velocity of robot in rad/s

R : radius of wheel in m

L : distance between wheels in m

Since a differential drive robot cannot move perpendicular to its current orientation, the v_y component of the velocity command is ignored.

2.2.2 Encoder based Odometry

The in-built encoders produce 360 ticks per revolution which amounts to a 1° resolution. Using these encoder readings, a *dead-reckoning* based approach is adopted to localize the robot. This is achieved using the following transformations:

$$\Delta x = \frac{R * (\Delta l + \Delta r)}{2} \tag{2.3}$$

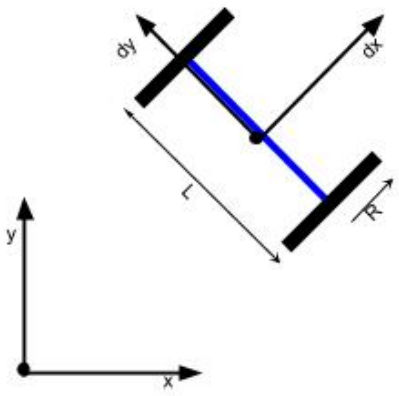
$$\Delta y = 0 \tag{2.4}$$

$$\Delta \theta = \frac{R * (\Delta l - \Delta r)}{L} \tag{2.5}$$

$$x \leftarrow x + \Delta x * cos(\theta) \tag{2.6}$$

$$y \leftarrow y + \Delta x * sin(\theta) \tag{2.7}$$

$$\theta \leftarrow \theta + \Delta \theta \tag{2.8}$$



A major limitation of the *dead-reckoning* approach with low resolution sensors is the continuous increase in error. Even a 1° error in the estimation of θ would lead to a large error when the robot moves forward by $1m$. In order to counter this, the vision based localization system is used to reset the positions periodically (say once in 2 seconds).

2.2.3 Sensors

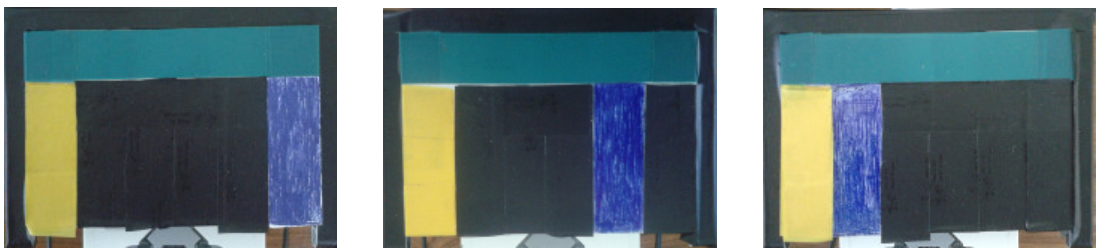
The LEGO EV3 kit comes with a variety of sensors including IR beacon, Ultrasonic Rangefinder, Sound, Light and contact sensors. These sensor information can be broadcasted through the network using ROS. An Ultrasonic rangefinder sensor is available for obstacle avoidance. The sensor measurements are *published* as a *LaserScan* message using ROS. The program closely resembles the example program used in the ROS tutorial - Writing a Simple Publisher and Subscriber (C++) and can be easily extended to other sensors. The **ev3-ros** package additionally has programs which enables the use of Color sensors and Contact sensors in order to increase the scope of usage.

2.3 Vision based Localization

Vision based localization of the different robots is achieved using coloured markers. The markers are designed to uniquely determine the identity of the robot and also determine its orientation.

2.3.1 Coloured Markers

The markers are made using strips of coloured tapes on a black background. Each marker has three colours - Green, Blue and Yellow. The relative positions of these strips is used in uniquely identifying the robot and also to estimate the orientation of the robot.



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

Figure 2.2: Three unique coloured markers made using strips of Green, Blue and Yellow tapes. Many more markers can be constructed in a similar fashion.

2.3.2 HSV vs. RGB color space

The default color space used for representing a pixel is RGB (Red Green Blue). But this representation poses difficulty for color based segmentation. An alternative is the HSV (Hue Saturation Value) color space which is suitable for color based segmentation.

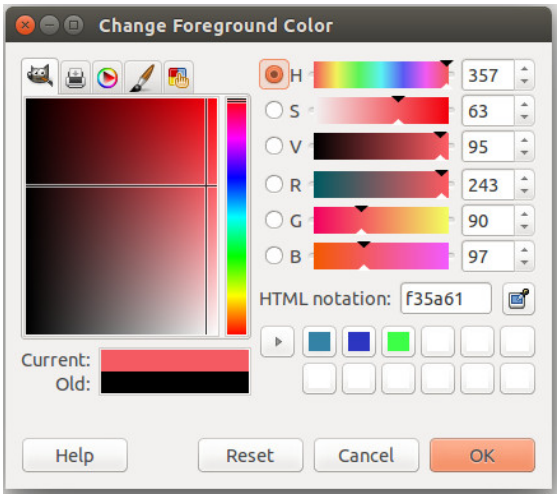


Figure 2.3: Colour selection dialog box from the GIMP software.

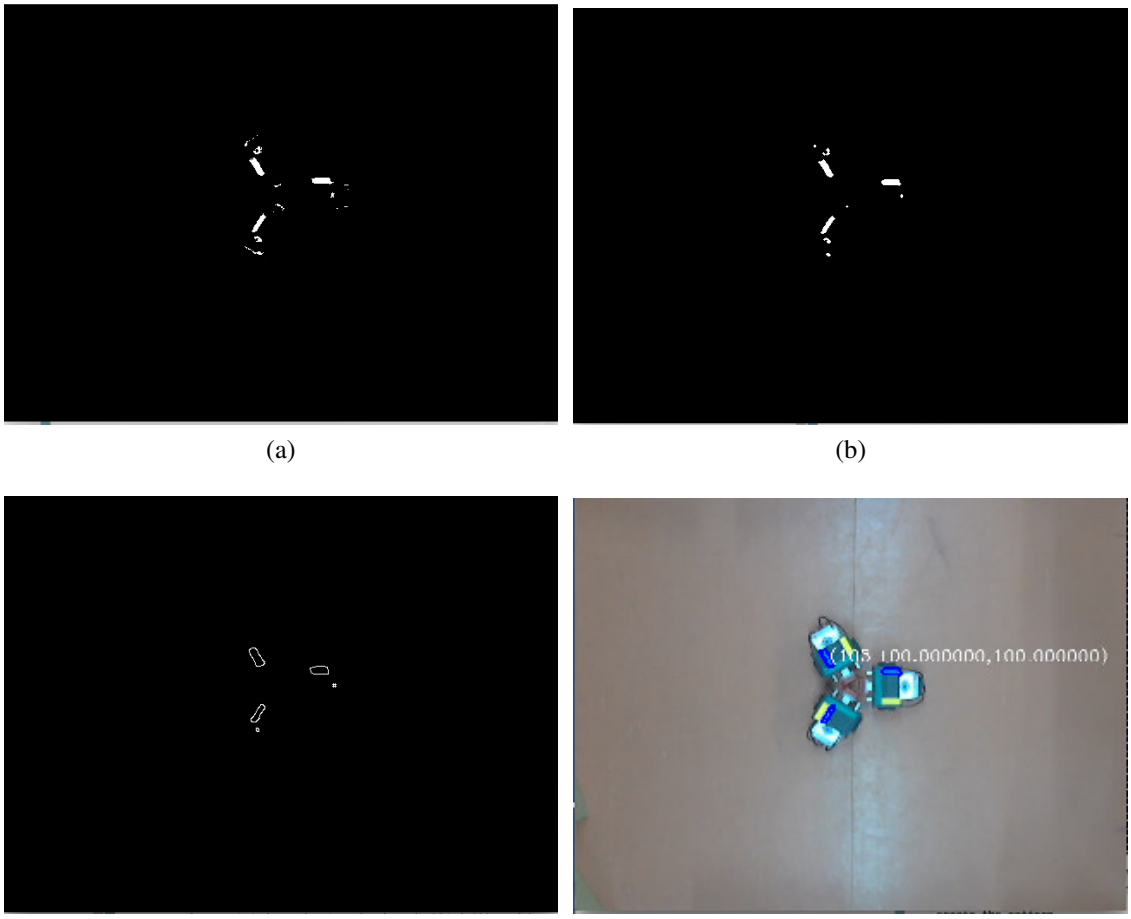
The 6 sliders in Fig.2.3 represent the colors that would result in changing the corresponding values. As can be observed, the HSV color space provides a clear grouping of the color so that various shades of the same colour differ only in the Saturation or Value component of the pixel while the RGB color space lacks this nice property. This color based segmentation is implemented using OpenCV library.

2.3.3 Colour based Segmentation

Algorithm 1 Color based segmentation

```
1:  procedure GETCONTOURS(frame)
2:      cvtColor(frame,BGR2HSV)
3:      inRange(frame,lower_limit,upper_limit)      ▷ The limits depend on colour
4:      apply MORPH_OPEN filter                     ▷ to remove pepper-and-salt noise
5:      detect edges using Canny detector
6:      find contours
7:      filter contours based on area
8:      calculate centroid of the contours
9:      return centroids
10: end procedure
```

Fig.2.4 shows the different steps in color based segmentation and Alg. 1 provides a pseudo code of the algorithm. Fig. 2.4a shows the frame after Step 3 in the GetCon-



(a) (b)

(c) (d)

Figure 2.4: a) Pixels belonging to blue family alone are marked white b) After applying MORPH OPEN filter c) Boundaries of computed contours d) Determined contours are overlayed on the initial input image

tours procedure, Fig. 2.4b shows the frame after Step 4 and Fig. 2.4c shows the frame after Step 7. The computed contours are outlined in red in Fig. 2.4d. Once the contours and their centroids are determined, the contours of different colours are grouped together based on distance. The three centroids in each group form a unique triangle using which the orientation of the robot is estimated. Due to noisy data, all the blobs might not be detected in every frame. As can be seen in Fig. 2.5, the yellow blob from one of the robot markers was not successfully detected and hence its overall position and orientation could not be determined.

Figure 2.5: The yellow blob on the coloured marked for the robot on the right has not been detected due to noisy data contributed by variations in lighting conditions and reflections from nearby regions.

2.4 Integration with ROS framework

Robot Operating System (ROS) is a popular library used by roboticists around the world. ROS provides a unified framework for developing and sharing Robotics software. All the major robot manufacturers provide drivers which are compatible with

ROS. Thus, the ability to interface the LEGO EV3 robots with ROS greatly expands the scope of using LEGO EV3 kits in research and education. The ROS framework provides a robust framework for communication between the various nodes (robots and/or PC) in the ROS network. In this project, three robot nodes and one PC node (which is connected to the camera) are used. More robots and/or PCs can be easily added to the same ROS network but only one node can act as the master node.

2.4.1 Sensor data visualization

A powerful feature of ROS is the Visualization tool called rviz that allows graphical visualization of the sensor data and robot movements. Fig. 2.6 shows the position of three robots and the respective ultrasonic range finder's readings. Refer ROS documentation on rviz for further details.

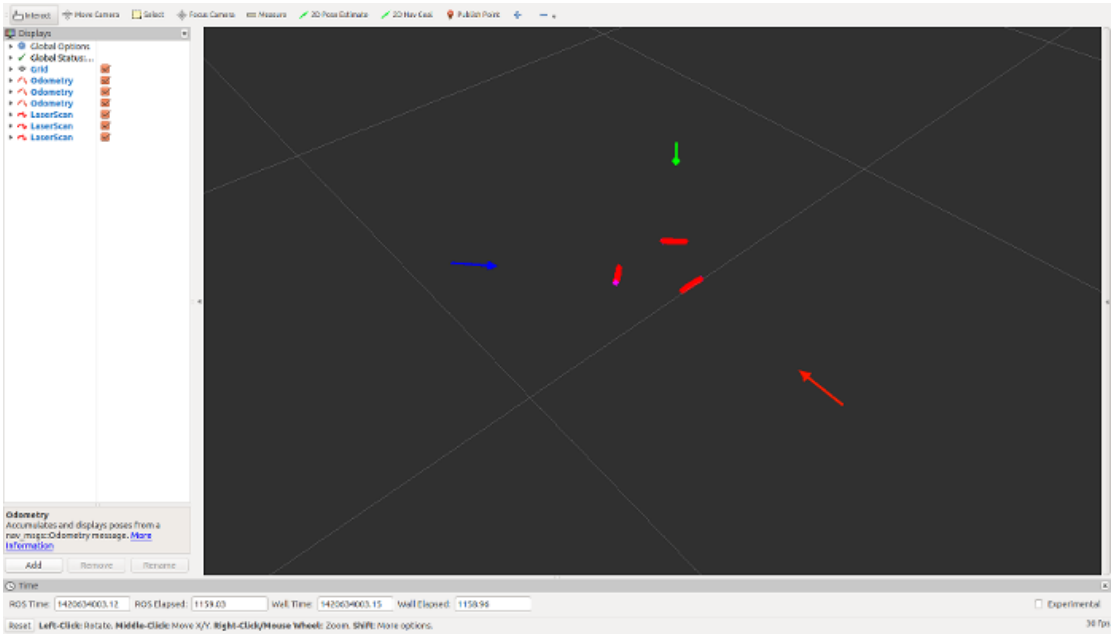


Figure 2.6: The pose of each robot is shown as a coloured arrow with its base at the position of the robot and the arrow head pointing along the orientation of the robot. A single obstacle placed at the center is detected by all three robots as can be seen by the red lines above.

2.4.2 Interfacing with LEGO EV3 using ROSSerial

ROSSerial is a ROS package that is used to communicate with each of the robots. This package allows a ROS network to communicate with embedded linux devices, arduinos

and other embedded systems. You can refer the ROS documentation for further details. Fig. 2.7 shows the ROS network for our system. The `rosserial_server` node behaves as a proxy between the PC node and the individual LEGO EV3 robot units. Hence to the main ROS network, all the information appears to be published and subscribed by `rosserial_server` node.

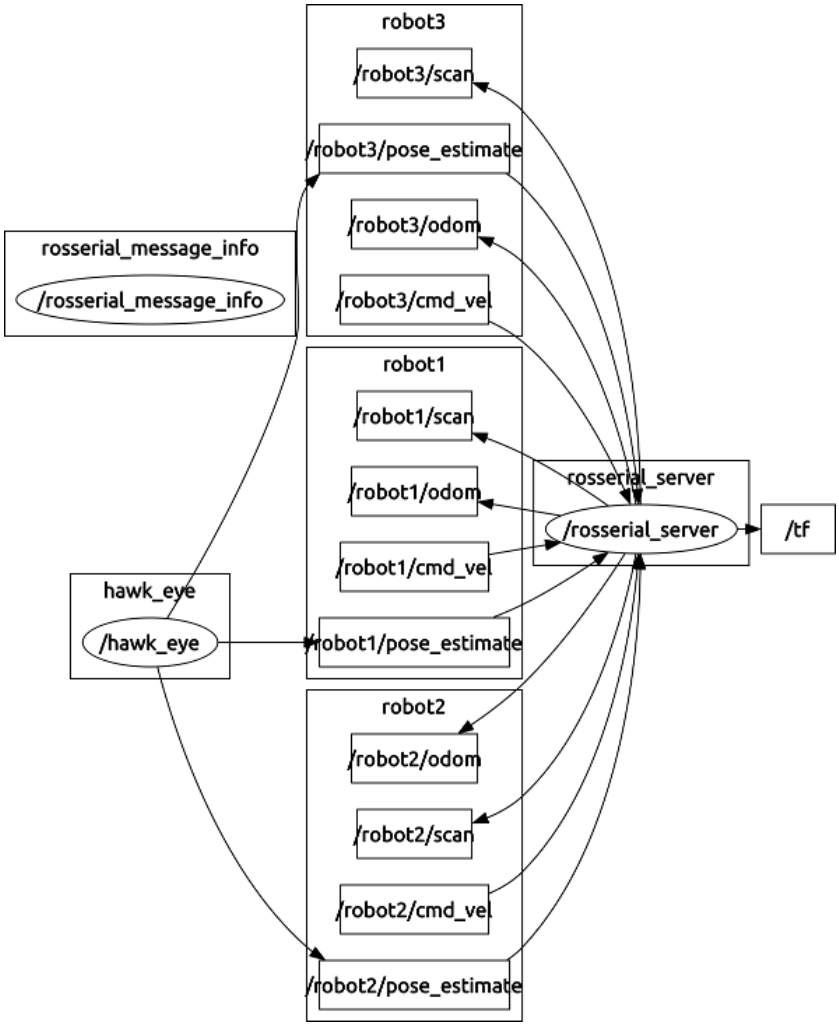


Figure 2.7: The state of the ROS system visualized using *rosgraph*

2.4.3 Keyboard/Joystick teleoperation

The teleop_twist_keyboard and teleop_twist_joystick packages available in ROS can be used to remotely control the individual lego robots via keyboard and joystick respectively. By default, these packages publish velocity commands via *cmd_vel* topic. But in order to individually control different robots in the same network, the velocity commands have to published in their respective namespace (i.e.) */robot1/cmd_vel*, */robot2/cmd_vel* and */robot3/cmd_vel*.

2.4.4 Random exploration with obstacle avoidance

Now that we are capable of controlling the three robots individually and also localizing the robots accurately as long as they remain within the field of view of the aerial camera, we can use this platform to test higher level algorithms. As a simple example, a *wanderer* script has been written. This script sends velocity commands to each of the robot based on a set of rules:

1. if an obstacle is present less than 20cm ahead, turn right
2. if another robot is nearby, move backward and turn right simultaneously
3. else move forward

Fig. 2.8 displays the odometry information of the three robots as they "wandered" based on the same simple rules. Apart from visualization, this data can be stored for further analysis using `rosbag`.

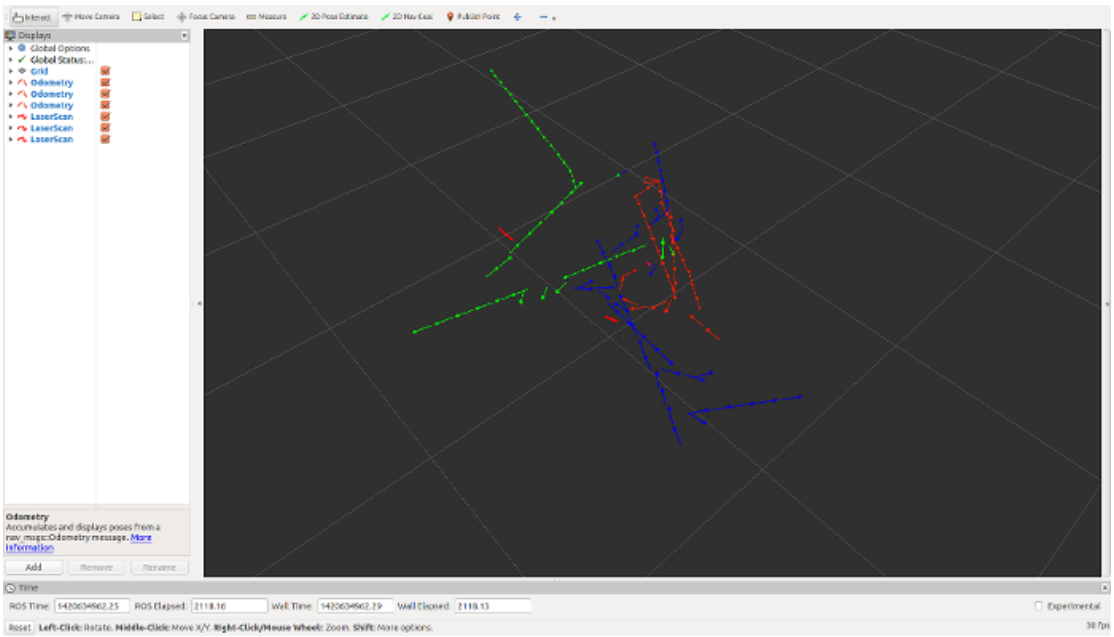


Figure 2.8: The path taken by the three robots are shown using arrow marks of their respective colours

2.5 Avenues for Future Work

The integration with ROS framework in itself greatly extends the scope of this platform. The project can be extended in various dimensions some of which have been listed below:

1. Built-in behaviors (move in an arc, move forward by distance, turn by angle etc. . .) for the differential drive robots available as ROS services.

2. Implementing tracking to improve accuracy and speed of the vision system

3. Adding a gripper mechanism to the robots to allow for testing manipulation problems

4. Replacing the stationary birds-eye view camera with a camera mounted on Quadrotor for use in outdoor environments.

CHAPTER 3

ISAACS TARGET GUARDING PROBLEM: AN EXPERIMENTAL STUDY

3.1 Target Guarding Problem

This section discusses the target guarding problem formulated as a differential game and the optimal strategies proposed by Raghav et. al. Note that this discussion is presented here for completeness and our work only deals with studying the performance of these policies on our test bed.

The objective of the opponent robot *O* is to enter the target area *T* without getting captured by the guard robot *G*. In other words, *O* attempts to minimize the distance *OT* at the end of the game while *G* attempts to maximize the same. The game ends if *O* enters *T* evading capture or if *G* captures *O* before *O* reaches *T*. For simplicity, assume that capture occurs if the two agents coincide. Let v_o and v_g denote the velocity of *O* and *G* respectively and $v_g > v_o$ unless stated otherwise. Let (x_o, y_o) be the current position of *O* and (x_g, y_g) be the current position of *G*. Now the 2D plane can be partitioned into two regions wherein each of the agent is able to reach any point in the region faster than the other agent. Assuming the agents to be point objects that can instantaneously move in any direction, the locus of the points that forms the boundary between these two regions is given by:

$$\frac{\sqrt{(x-x_o)^2+(y-y_o)^2}}{v_o} = \frac{\sqrt{(x-x_g)^2+(y-y_g)^2}}{v_g} \tag{3.1}$$

In other words, Eqn. 3.1 represents the set of points for which both agents would take the same amount of time to reach. Let $x_c = \frac{x_o v_g^2 - x_g v_o^2}{v_g^2 - v_o^2}$ and $y_c = \frac{y_o v_g^2 - y_g v_o^2}{v_g^2 - v_o^2}$. Then, Eqn. 3.1 becomes

$$(x-x_c)^2 + (y-y_c)^2 = R^2 \tag{3.2}$$

where $R = \sqrt{x_c^2 + y_c^2 - \frac{v_g^2(x_o^2+y_o^2) - v_o^2(x_g^2+y_g^2)}{v_g^2 - v_o^2}}$. As shown in Fig. 3.1, Eqn. 3.2 represents

a circle, called the *capture circle C*, encompassing *O*. Note that if $v_g < v_o$, then the circle would be encompassing *G* instead.

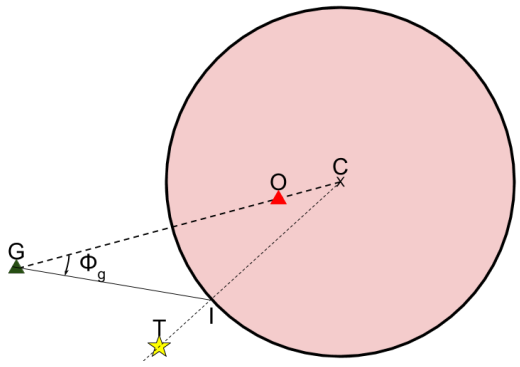


Figure 3.1: The red and green triangles denote the positions of *O* and *G* respectively. The yellow star represents *T* and its closest point on the capture circle is *I*. The shaded region is reachable by *O* faster than *G*.

If the initial position of *T* is inside *C*, then *O* can always reach *T* while evading capture by *G* if it follows *pure-pursuit* strategy (i.e.) move directly towards *T*. Therefore, we do not investigate this case any further and focus on the case where the initial position of *T* is outside *C*. In this scenario, the optimal strategy for both agents is to move to the point *I* which is the closest point of *T* on *C*. If either of the agents deviate from this trajectory, it plays to the favour of the other agent. This is, in essence, the defining trait of a differential game.

Following Pontryagin's Maximum Principle, both agents travel at their maximum velocities v_o and v_g respectively. Therefore, the problem is reduced to determining the optimal heading θ_g for *G* given the current positions of $G(x_g, y_g)$, $O(x_o, y_o)$ and $T(x_t, y_t)$. In order to allow for a real-time implementation of the optimal strategy, a fast computation method for calculating θ_g is presented below.

$$k_0 = \frac{\|\vec{GT}\|}{\|\vec{GO}\|} \tag{3.3}$$

$$k_1 = \frac{v_o}{v_g} \tag{3.4}$$

$$k_2 = k_0(1 - k_1^2) \tag{3.5}$$

$$\psi = \angle OGT \tag{3.6}$$

$$k_3 = k_2^2 - 2k_2 \cos\psi + 1 \tag{3.7}$$

$$\phi_g = \sin^{-1}\left(\sqrt{\frac{k_1^2 k_2^2 \sin^2\psi}{k_3[k_1^2 - \frac{2k_1\|k_2\cos\psi - 1\|}{\sqrt{k_3}} + 1]}}\right) \tag{3.8}$$

$$\theta_g = \angle\vec{GO} - \phi_g \tag{3.9}$$

3.2 Extending the Testbed

The test platform presented in Chapter 2 uses an RGB camera to detect specially designed coloured markers to compute the position and orientation of the individual robots. Adequate performance of such a system requires maintaining optimal lighting conditions and absence of reflecting surfaces. The distortions present in the captured image results in a noisy estimate for the position and orientation of the robots. In order to counter these short-comings, a new localization method was developed which uses a depth sensor such as Microsoft Kinect to accurately determine the position of the robots. The orientation of the individual robots are measured using gyroscope sensors available in the LEGO Mindstorms EV3 kits.

3.2.1 Experimental Setup

The experimental setup consists of a depth sensor, such as kinect, mounted from the ceiling at a known height H_{cam} above the ground plane and facing vertically down as shown in 3.2. The Kinect for Xbox One sensor has a horizontal field of view of $70°$, a vertical field of view of $60°$ and a range of 4.5m. The value of H_{cam} is then chosen based on the amount of ground area that needs to be covered. We assume that there are no obstacles within the field of view of the sensor and the robots are the only objects

present. If suppose the field contains other objects, the size of the object can be used to differentiate the robots from others.

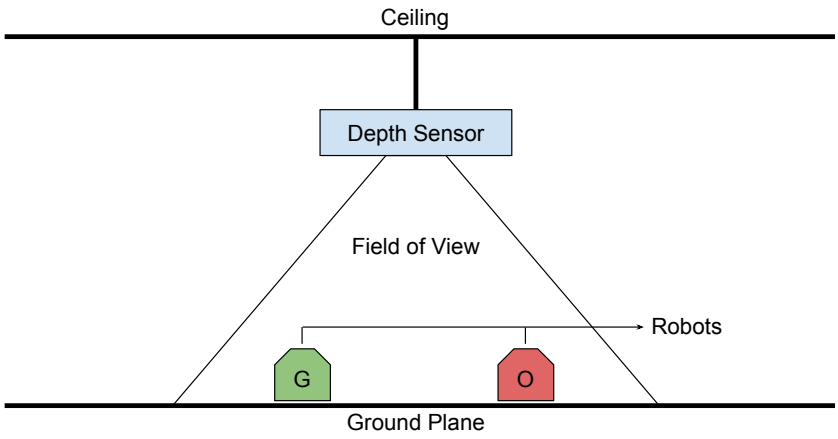


Figure 3.2: Sectional view of the new test bed setup

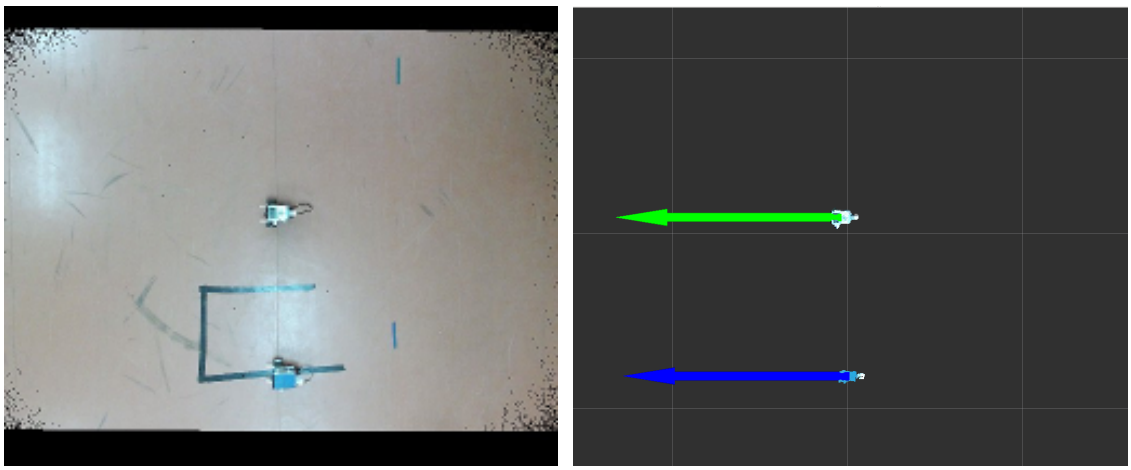
3.2.2 Localization algorithm

Algorithm 2 Depth based localization

```
1:  procedure GETPOSE(cloud_full)
2:      cloud ← PCLremoveGroundPlane(cloud_full)
3:      clusters ← PCLeuclideanClusterExtraction(cloud)
4:      for blob in clusters do
5:          centroid ← PCLget3DCentroid(blob)
6:          id ← determineID(blob)
7:          heading ← getHeading(id)
8:          pose ← constructPoseMsg(centroid,heading)
9:          global_pose ← transformToGlobalFrame(pose)
10:         ros.publish(global_pose, id)
11:     end for
12: end procedure
```

The pseudo-code to determine the position and orientation of the individual robots is shown in Algorithm 2. The input to the algorithm is the current point cloud produced by the depth sensor, see Fig. 3.3a. We use the Point Cloud Library (PCL) (Rusu and Cousins, 2011) for fast and efficient processing of these point clouds. First, the ground plane is removed by applying a conditional filter such that only points that satisfy $z < H_{cam}$ are retained, where H_{cam} is the height above the ground plane at which the depth sensor is mounted. Next, on line 3, we apply a clustering algorithm which partitions the point cloud such that any two points that are within δ distant from each other belong to the same partition (or cluster). As Kinect provides a dense point cloud, the typical

value for δ is taken to be 5cm. Note that in the absence of obstacles, each of the cluster corresponds to an individual robot. The clusters are then processed in sequence. On line 6, the RGB values of the points in the cluster is used to identify the robot. Since there are only two robots in our application, we use a light colour for *O* and a dark colour for *G*. Each robot publishes its current heading, measured using a gyroscope sensor, on separate topics in the ROS network. We obtain the heading of the robot by listening to the appropriate topic based on the id of the robot. Finally, the complete 2D pose, constructed on line 8, is transformed to the global frame using the known static pose of the depth sensor. This message is then broadcast through the ROS network on the topic corresponding to the id of the robot. Fig. 3.3b shows the computed pose vectors for both the agents as shown in *rviz*.



(a) (b)

Figure 3.3: a) Registered pointcloud that is the input to the algorithm. b) Extracted pointcloud clusters of the robots along with their pose vectors

3.3 Experiments

Recall that, the optimal strategies used in this work have been derived assuming a point robot that can instantaneously move along any direction (i.e.) a 2D holonomic robot. But most robots in real world are non-holonomic such as the differential drive configuration of the LEGO EV3 robots used in this study. These robots require additional time to orient themselves along the desired direction of movement. Here, we use a simple proportional controller for each robot as shown in Fig. 3.4 to maintain the desired orientation of the robots. For simplicity, we assume the robots can only move forward or

turn but cannot move in reverse.

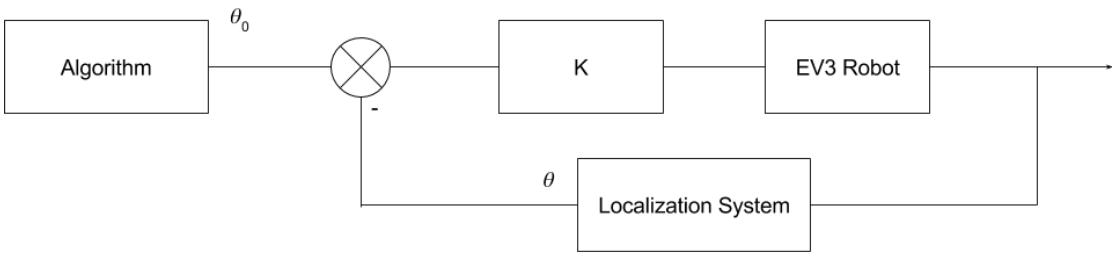


Figure 3.4: A proportional controller for the robot to maintain desired orientation

The initial time required to face along the desired angle could potentially shift the game to the opponent's favour or vice versa. In order to analyse this, the following four experiments with different starting conditions were performed. In all these experiments, the motion of *G* was controlled by the algorithm and *O* was manually controlled via keyboard.

- Case A
  G is aligned along $\vec{GI}$, *O* is aligned along $\vec{OT}$ and *T* is outside *C*
- Case B
  G is aligned along $\vec{GI}$, *O* is anti-aligned to $\vec{OT}$ and *T* is inside *C*
- Case C
  O is aligned along $\vec{OT}$, *G* is anti-aligned to $\vec{GI}$ and *T* is outside *C*
- Case D
  A pure-pursuit scenario where *G* is facing along *O*, *O* is facing along *T* and *T* is outside *C*

3.3.1 Case A

This experiment aims to validate the optimal policies for the autonomous guard robot. Therefore, to nullify the effects of the differential drive constraints, both the agents are initially aligned along their respective destinations. As a result, the trajectory followed by both agents is a straight line as shown in 3.5. Further, the red plot in Fig. 3.5 shows the optimal trajectory for *G* that is computed using simulations (assuming a 2D holonomic robot) for the trajectory of *O* obtained from the experiment. It is clear that the experimental results agree with the simulations.

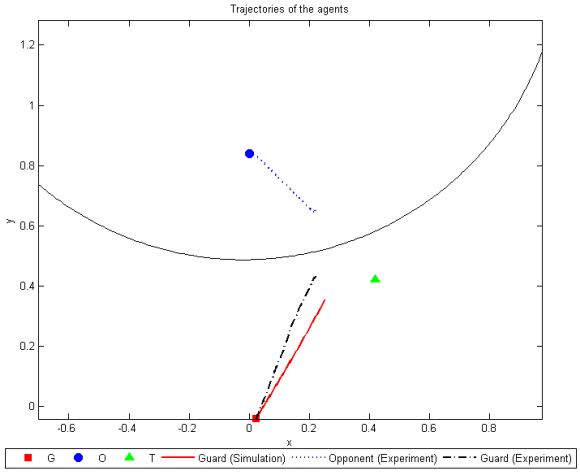


Figure 3.5: G successfully intercepts O before O captures T

3.3.2 Case B

In this experiment, T is initially inside C. Therefore, O will win the game following pure-pursuit strategy if it had been a 2D holonomic robot. On the contrary, the experiment portrays a scenario where G is able to protect T as it is initially aligned along $\vec{GI}$ and the time taken by O to turn around is sufficient for G to pull T inside C through the course of the game. Note that C continuously moves as the game progresses. In summary, the initial orientations of the agents shifts the game in favour of G.

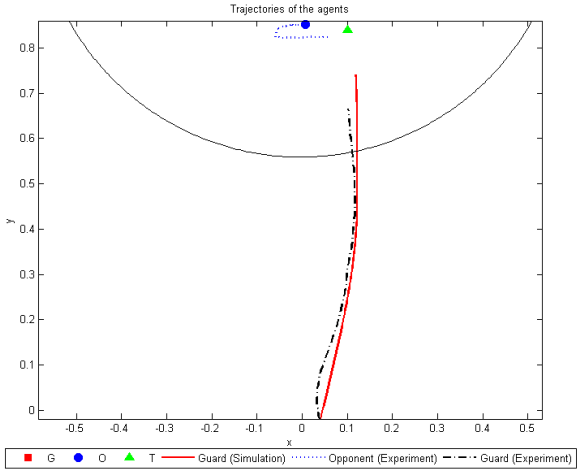


Figure 3.6: G wins even though T is initially inside C due to the time taken by O to turn around.

3.3.3 Case C

In this experiment, G is at a disadvantage as it is initially anti-aligned to $\vec{GI}$. But simulations show that G will win the game and intercept O. On the contrary, the experiment portrays a scenario where O is able to capture T as it is initially aligned along $\vec{OT}$ and the time taken by G to turn around is sufficient for O to reach T. In summary, the initial orientations of the agents shifts the game in favour of O.

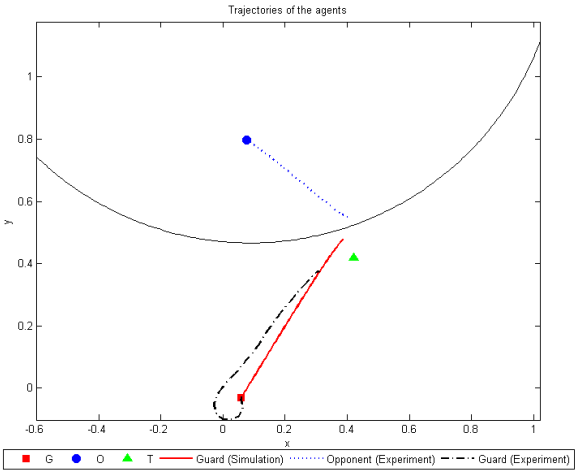


Figure 3.7: O wins even though T is initially outside C due to the time taken by G to turn around.

3.3.4 Case D

This experiment shows the progress of the game when O plays sub-optimally by following pure-pursuit strategy. As shown in Fig. 3.8, G is able to win the game. But the initial error in the orientation of G shows that G performs better in simulation than in the experiment.

3.4 Conclusion and Future work

Optimal strategies for the target guarding problem have been presented in literature (Venkatesan and Sinha, 2014). But these analytical solutions do not take into consideration the constraints imposed by the specific dynamics of real robots. In this work, we have studied the performance of these analytic solutions using differential drive

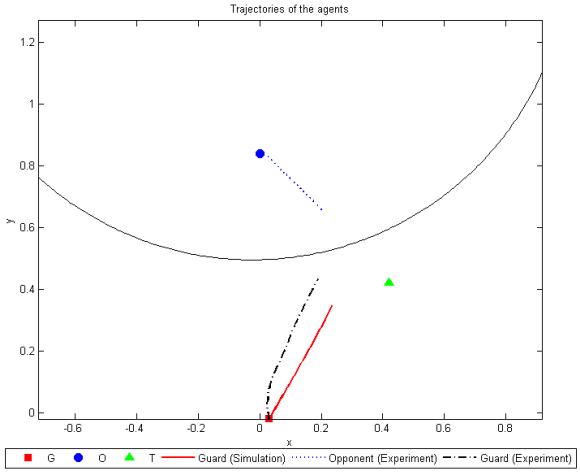


Figure 3.8: *G* successfully protects *T*

robots. We have showed that the optimal strategies perform well if the robots are initially aligned along the desired direction of movement prescribed by the algorithm. In addition, we have also demonstrated two scenarios where the initial mis-alignment of the robots shifts the game in favour of the opponent or vice versa. Though the analytic solutions can be implemented in real-time on robots with differential drive constraints, these are not necessarily the optimal solutions. Therefore, further work is required to derive optimal strategies that consider the dynamics of the participating agents. Finally, the extended testbed using depth based localization serves as a useful tool to study a variety of games in the broad field of pursuit-evasion. Future work includes developing optimal strategies that consider the vehicle dynamics and also the presence of obstacles in the path of the robots.

Part II

Autonomous Mapping of Unknown Structures

CHAPTER 4

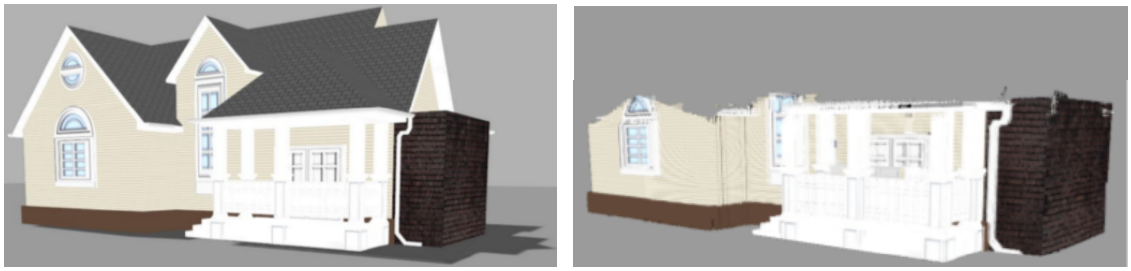
Motion Planning Strategies for Autonomous Mapping

4.1 Introduction

Accurate 3D computer models of large structures have a wide range of practical applications, from inspecting an aging structure to providing virtual tours of cultural heritage sites (Jacobi, 2015; El-Hakim *et al.*, 2004). In order to autonomously build such a 3D model in real-time, we need to address two problems. First, we need a robust mapping system that can build the 3D model on the fly when given a sequence of images or depth maps as input. This is a widely researched problem called Visual Simultaneous Localization and Mapping (vSLAM), for which several open source packages offer increasingly accurate and efficient solutions (Labbe and Michaud, 2014; Endres *et al.*, 2014). The second problem relates to active sensing (Bajcsy, 1988), as we need motion planning strategies that can guide a mobile sensor to explore the structure of interest. For mapping, monitoring or inspection applications, certain classical strategies such as frontier-based exploration algorithms (Yamauchi, 1997), which guide the robot to previously unexplored regions irrespective of whether it is part of the structure of interest or not, are not necessarily well adapted.

The goal of this work is to guide a mobile ground robot equipped with a depth sensor, in order to autonomously determine the boundaries of an initially unknown structure, build a 3D model of the structure and attempt to fill holes in the model so that the reconstruction is as accurate and complete as possible. Some recent work considers the problem of reconstructing a 3D model of arbitrary objects by moving a depth sensor relative to the object (Kriegel *et al.*, 2015; Krainin *et al.*, 2011). Typically, these systems iteratively build a complete 3D model of the object by heuristically choosing the next best viewpoint according to some performance measure. However, much of this work is restricted to building models of relatively small objects that are bounded by the size of the robot workspace. In contrast, our focus is on 3D reconstruction of larger but still bounded structures such as buildings, which can be several orders of magnitude larger than a mobile robot. The related problem of automated inspection deals

with large structures such as tall buildings (Lin *et al.*, 2015) and ship hulls. Bircher et al. (Bircher *et al.*, 2015) assume that a prior 3D mesh of the structure to inspect is available and compute a short path connecting viewpoints that together are guaranteed to cover all triangles in the mesh. As they point out, the inspection problem starting from a prior model is related to coverage path planning, see, e.g., (Acar *et al.*, 2006; Lim *et al.*, 2014). In (Englot and Hover, 2012), Englot et al. begin by assuming a safe bounding box of the hull and construct a coarse mesh of the hull by tracing along the walls of this box in a fixed trajectory without taking feedback from the actual geometry of the structure. Moreover, this coarse mesh is manually processed offline to yield an accurate 3D mesh which is then used to inspect the finer structural details. Sheng et al. (Sheng *et al.*, 2008) use a prior CAD model of an aircraft to plan a path for a robotic crawler such that it inspects all the rivets on the surface of the aircraft. In this work however, we do not assume any prior information in terms of a 3D mesh, CAD model or a bounding box around the structure, and focus on reactive path-planning to build the model online.



(a) (b)

Figure 4.1: Comparison of the a) Simulated Model in Gazebo (Andrew Howard) that needs to be mapped and b) Reconstructed 3D model by a mobile ground robot using our policies. Only the bottom portion is mapped due to the limited reachable space of the sensor.

In computer vision and photogrammetry, Structure from Motion (SfM) techniques aim at building a 3D model of a scene from a large number of images (Frahm *et al.*, 2010; Wu, 2011; Furukawa and Ponce, 2010). But most of this work focuses on batch post-processing and in any case assumes a given dataset. On the other hand, our work focuses on actively exploring the environment to build a complete model in real-time, with our controller taking at any time the current model as an input. Naturally, eventual completeness of the model can be limited by the physical characteristics of the robot, and specifically the reachable space of the sensor, see Fig. 4.1. We emphasize that we do not discuss in details the task of actually building the model from the collected depth

maps, which can be executed by one of the available vSLAM systems, such as the Real-Time Appearance Based Mapping package (RTAB-Map) (Labbe and Michaud, 2014) that we use in our simulations. This package can in fact be replaced with little change to our algorithms by any vSLAM system based on pose-graph optimization (Kuemmerle *et al.*, 2011). State-of-the-art SfM systems can also be used to post-process the sequence of images or depth maps captured using our policies in order to obtain a more accurate model.

Finally, another line of work in informative path planning relates to autonomous exploration and coverage of relatively large environments, using variants of frontier based exploration algorithms for example (Shade and Newman, 2011; Shen *et al.*, 2012; Heng *et al.*, 2015; Atanasov *et al.*, 2015). While these papers focus on path planning to quickly build models of potentially large and complex spaces, they do not address the problem of autonomously delimiting and mapping as completely as possible a specific bounded structure of interest.

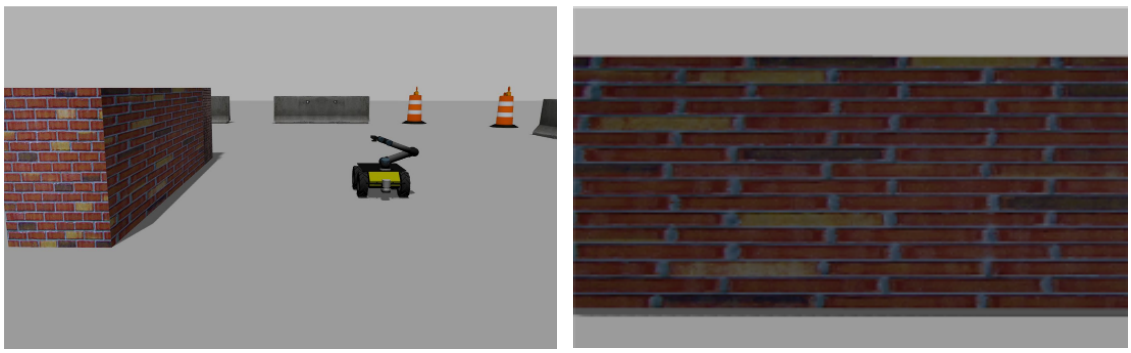
In summary, the key contributions of this work are:

- a motion planning strategy to autonomously determine the boundaries of an unknown structure using a ground robot;
- a novel algorithm to determine incomplete portions of the partially constructed model;
- motion planning policies for automatically exploring and adding these missing portions to the model;
- and an evaluation of the proposed policies via simulations.

The rest of this chapter is organized as follows. We begin with a detailed presentation of the problem in section 4.2. In section 4.3 we present our policies for autonomously determining the boundaries of the unknown structure. Section 4.4 describes algorithms for detecting the missing portions and completing the model. In section 4.5, we evaluate the proposed policies via simulations and present a comparison with the classic frontier based exploration algorithm. Finally, we discuss avenues for future work and conclude in section 4.6.

4.2 Problem Statement and Assumptions

Consider the problem of constructing a 3D model of a given structure of finite size, such as a monument or a building for example. Initially, no approximate model of the structure nor map of the environment is available, and the actual size of the structure is also unknown. We address the following question: "How should a mobile robot carrying a depth sensing camera, such as a Kinect, move in order to reconstruct a complete 3D model of this structure?". The sensor collecting depth and luminance images of the scene allows the robot to build the 3D model on the fly using available SLAM algorithms, such as RTAB-Map Labbe and Michaud (2014) or RGBD-SLAM Endres *et al.* (2014). These algorithms assemble the sequence of points clouds captured by the sensor (also called camera in the following), producing a registered global point cloud or a 3D occupancy grid stored in an OctoMapHornung *et al.* (2013). Note that the type of sensor (monocular camera, stereo camera rig with IMU, etc...) used depends on the SLAM algorithm. Any of these algorithms can be used with our policies as long as the SLAM module can additionally return the sequence of point clouds and corrected camera positions following registration. The remaining problem that we consider here is to determine the trajectory of the camera such that the entire visible portion of the structure is eventually captured in the model. The key challenge is to develop strategies that are applicable for any type of structure while respecting the physical limitations of the platform.



(a) (b)

Figure 4.2: The starting configuration of the robot needs to satisfy Initial Conditions 1 and 2. a) At the beginning, the structure and the robot can be separated by a plane. b) The initial image as seen by the camera. Initially, the robot only knows that the structure in the FOV of its camera is the one that should be mapped.

In order to specify to our system which structure is to be mapped, we require that the initial configuration of the robot with respect to the structure satisfy the following two basic conditions, illustrated in Fig. 4.2.

Initial Condition 1 *The robot is placed fully outside the structure, so that there exists a 2D plane separating the robot and the structure.*

Initial Condition 2 *The structure to be mapped is present in the field-of-view (FOV) of the camera.*

Next, note that depending on the set of all configurations that are reachable by the mobile sensor mounted on a specific physical platform, some parts of the structure might not be visible at all, and hence cannot be mapped by any algorithm implemented on this platform. For concreteness, we make the following assumption to describe our scenario and algorithms, but other situations could be handled with the generic tools developed in this paper.

Assumption 1 *The camera is mounted on a mobile ground robot such that the camera center* C *lies directly above the center* R *of the robot's base Frame of Reference (FoR), at a constant height. Moreover, the relative pitch and roll of the camera with respect to the robot FoR are kept fixed, while the relative yaw is unconstrained.*

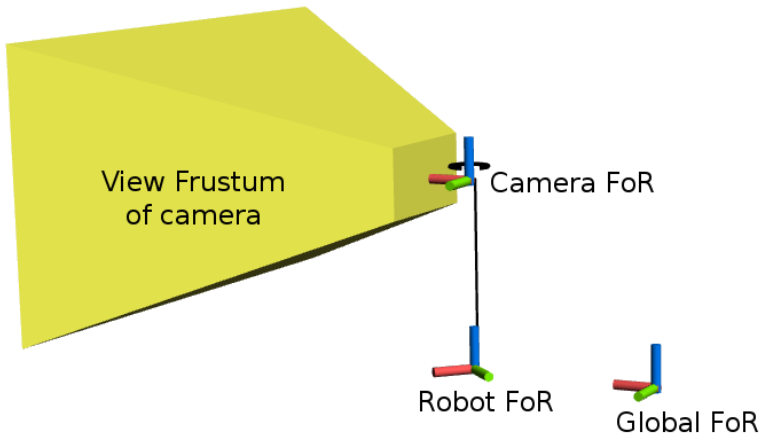


Figure 4.3: The camera is kept at a constant height above the robot's base. The red, green and blue lines correspond to the x, y and z axes respectively and both the camera and robot can rotate along their z axes. The yellow region corresponds to the view frustum of the camera.

Based on Assumption 1, Fig. 4.3 shows our conventions for the different FoR used. The global point cloud is assembled in a global FoR $\mathsf{G}\mathbf{x}_\mathsf{g}\mathbf{y}_\mathsf{g}\mathbf{z}_\mathsf{g}$. Note that we use bold font to represent vectors. The robot FoR $\mathsf{R}\mathbf{x}_\mathsf{r}\mathbf{y}_\mathsf{r}\mathbf{z}_\mathsf{r}$ (forward, left, up) is attached to a point R on the base of the mobile robot and moves along with it. The camera FoR $\mathsf{C}\mathbf{x}_\mathsf{c}\mathbf{y}_\mathsf{c}\mathbf{z}_\mathsf{c}$, with $\mathbf{z}_\mathsf{c} = \mathbf{z}_\mathsf{r}$, is rigidly attached to the robot except for the yaw motion, which is left unconstrained. The imaging plane of the camera is defined by $\mathbf{y}_\mathsf{c}\mathbf{z}_\mathsf{c}$, with $\mathbf{x}_\mathsf{c}$ pointing towards the front of the camera on the optical axis. Coordinates in the camera, robot and global FoR are denoted using superscripts as $\mathbf{v}^\mathsf{c}$, $\mathbf{v}^\mathsf{r}$ and $\mathbf{v}^\mathsf{g}$ respectively for a vector $\mathbf{v}$.

We make two additional assumptions for simplicity of exposition. First, let ρ be the maximum distance that we wish to allow between the structure and the camera when capturing point clouds. This distance could be the range of the camera or a shorter distance for which the resolution is higher. The next assumption guarantees that there exists collision free paths around the structure.

Assumption 2 *The distance of the closest obstacle from the structure is at least* 2ρ*.*

The next assumption simplifies the problem of detecting, tracking and removing the ground surface from point clouds.

Assumption 3 *The structure and the robot are placed on a horizontal surface (so* $\mathbf{z}_\mathsf{c} = \mathbf{z}_\mathsf{r} = \mathbf{z}_\mathrm{g}$*).*

A consequence of these assumptions is that relatively horizontal surfaces that are at the same height or above the camera center cannot be mapped, and the maximum height of the structure that can be mapped is $H_{\max} = z^\mathsf{g}_\mathsf{C} + \rho \tan\psi/2$, where ψ is the vertical angle of view of the camera and z^g_C the height of the camera. Assumption 3 could be removed by using recent classification systems that can differentiate between ground and non-ground regions (Zhou *et al.*, 2012) to pre-process the point clouds before sending them to our system.

Finally, there are additional implicit assumptions that we state informally. First, since we rely on an external mapping module to build the 3D model, the conditions that allow this module to operate sufficiently reliably must be met. For example, vSLAM generally requires appropriate scene illumination and the presence of a sufficiently rich

set of visual features. Second, we concentrate on the reconstruction of the details of the model at a scale comparable with or larger than the typical length of the robot. If features at a smaller scale need to be included, e.g., fine structural details on a wall, our system could be augmented with a more local planner for a robotic arm carrying the sensor (Kriegel *et al.*, 2015; Dornhege and Kleiner, 2013), as well as targeted computer vision techniques (Furukawa and Ponce, 2010). Finally, for reasons explained in Section 4.3.3, we assume that the robot is equipped with sensors capable of detecting obstacles in a 180° region ahead of it and within a distance of ρ.

We divide our mapping process into two phases. The first phase is the Perimeter Exploration (PE) phase, during which the robot moves clockwise around the structure to determine its boundaries. The robot continuously moves towards previously unseen regions of the structure, with the exploration directed towards finding the limits of the structure rather than closely following its geometry. The PE phase ends when our algorithm detects that the robot has returned to the neighborhood of its starting point O and the vSLAM module detects a global loop closure. After completing the PE phase, the system determines the locations of potential missing parts in the constructed 3D model. We can then start the second phase, which we call the Cavity Exploration (CE) phase, during which the system explores these missing parts in the model. The following subsections explain each step of our process in detail.

4.3 Perimeter Exploration

In this chapter, we present our first contribution - a method to autonomously determine the boundaries of an unknown structure. From Assumptions 2 and 3, $z^{\mathrm{g}} = 0$ and $z^{\mathrm{g}} = H_{\max}$ are bounding horizontal planes for the model. The remaining problem is to determine the expansion of the structure in the $\mathbf{x}_{\mathrm{g}}\mathbf{y}_{\mathrm{g}}$ plane. To do this, the robot moves clockwise around the structure by determining online a discrete sequence of successive goals or waypoints. It tries to keep the optical axis of the depth sensor approximately perpendicular to the structure, which maximizes the depth resolution at which a given portion of the structure is captured, and increases the density of captured points. It also tries to maintain the camera center C on a smooth path at a fixed distance from the structure.

Algorithm 3 Algorithm for computing the next goal for the camera using the current point cloud in camera FoR.

1: **function** COMPUTENEXTGOAL(cloud_full)
2: cloud $\leftarrow$ PCLremoveGroundPlane(cloud_full)
3: cloud_slice $\leftarrow$ filterForwardSlice(cloud)
4: $\overline{p}^{\mathsf{c}} \leftarrow$ PCLcompute3Dcentroid(cloud_slice)
5: $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3; \lambda_1, \lambda_2, \lambda_3] \leftarrow$ PCA(cloud_slice)
6: $\tilde{\mathbf{n}} \leftarrow \mathbf{v}_3 - (\mathbf{v}_3 \cdot \mathbf{z}_{\mathsf{c}})\mathbf{v}_3$ ▷ Projection on the $\mathbf{x}_{\mathsf{c}}\mathbf{y}_{\mathsf{c}}$ plane
7: $\mathbf{n} \leftarrow \tilde{\mathbf{n}}\ \mathrm{sign}(\tilde{\mathbf{n}} \cdot \overrightarrow{\mathsf{C}\overline{p}^{\mathsf{C}}}); \mathbf{n} \leftarrow \mathbf{n}/\|\mathbf{n}\|$
8: $\mathbf{r} \leftarrow \mathbf{z}_{\mathsf{c}} \times \mathbf{n}$
9: $goal \leftarrow \overline{p}^{\mathsf{c}} - D\,\mathbf{n} + step\,\mathbf{r}$
10: **return** $goal, \mathbf{n}$
11: **end function**

4.3.1 Determination of the next goal

The pseudo-code to determine the next position and orientation of the camera in our perimeter exploration algorithm is shown in Algorithm 3. The algorithm takes as input the current point cloud produced by the camera in its FoR. For its implementation we rely on the Point Cloud Library (PCL) (Rusu and Cousins, 2011). First, the ground plane is removed so that the resulting point cloud $\mathcal{P}$ contains only those points that belong to the structure. Next, on line 3, we select a subset $\mathcal{S}$ of the point cloud referred to as the forward slice, which adjoins the part of the structure that must be explored next, see Fig. 4.4. Concretely, we choose $\mathcal{S}$ so that its y^{c}-coordinates satisfy $y^{\mathsf{c}}_{\max} - \frac{y^{\mathsf{c}}_{\max} - y^{\mathsf{c}}_{\min}}{3} \leq y^{\mathsf{c}} \leq y^{\mathsf{c}}_{\max}$, where $y^{\mathsf{c}}_{\min}$ and $y^{\mathsf{c}}_{\max}$ are the minimum and maximum y^{c}-coordinate values for all points in $\mathcal{P}$. On line 5, following (Mitra *et al.*, 2004), we compute via Principal Component Analysis (PCA) the normal direction to that plane Π which best fits $\mathcal{S}$. In more details, denote $\mathcal{S} = \{p^{\mathsf{c}}_i : i = 1, 2, \ldots, m\}$ and define the covariance matrix $\mathbf{X} = \frac{1}{m}\sum_{i=1}^{m}(p^{\mathsf{c}}_i - \overline{p}^{\mathsf{c}})(p^{\mathsf{c}}_i - \overline{p}^{\mathsf{c}})^T$, where $\overline{p}^{\mathsf{c}} = \frac{1}{m}\sum_{i=1}^{m} p^{\mathsf{c}}_i$ is the centroid of $\mathcal{S}$ computed on line 4. We compute the eigenvectors $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ of $\mathbf{X}$, ordered here by decreasing value of the eigenvalues $\lambda_1, \lambda_2, \lambda_3$. The eigenvector $\mathbf{v}_3$ for the smallest eigenvalue corresponds to the normal to the plane Π.

The algorithm returns $\mathbf{n}$, computed from the projection of the normal vector $\mathbf{v}_3$ on the $\mathbf{x}_{\mathsf{c}}\mathbf{y}_{\mathsf{c}}$ plane, and taken to point in the direction of the vector $\overrightarrow{\mathsf{C}\overline{p}^{\mathsf{C}}}$. This vector $\mathbf{n}$ defines the desired orientation of the camera. The algorithm also returns the next goal point $goal = \overline{p}^{\mathsf{c}} - D\,\mathbf{n} + step\,\mathbf{r}$ for the center C of the camera, where $D < \rho$ is the desired distance between the camera and the structure, $\mathbf{r} = \mathbf{z}_{\mathsf{c}} \times \mathbf{n}$ is computed on line

8, and $step = \frac{y^c_{\max} - y^c_{\min}}{6}$. The term $step\,\mathbf{r}$, which is along the plane Π, is used to shift the $goal$ forward so that both sections of a corner fall in the FOV of the camera, as in the situation shown on Fig. 4.4. This prevents the algorithm from making slow progress around corners. Finally, the computed camera pose is transformed into the global FoR to obtain the goal point g^{g} for the camera center C. We simplify the notation g^{g} to g in the following, where we work in the global reference frame.

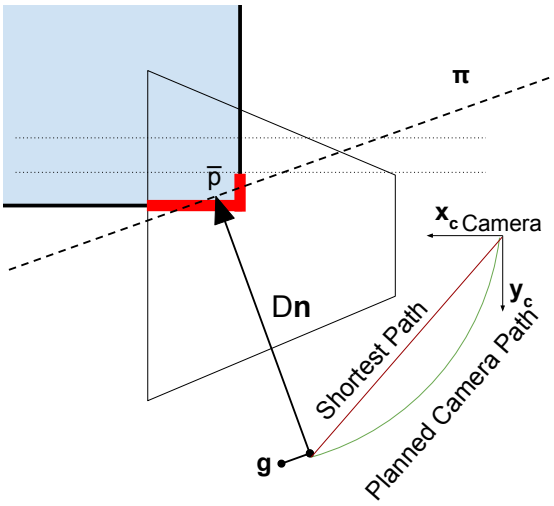


Figure 4.4: Top-down view illustrating the computation of the next goal. We show here the situation for a corner section of the structure. The forward slice $\mathcal{S}$ is highlighted in red and the corresponding best fit plane Π is shown as well.

4.3.2 Local path planning to the next goal

In order to move the camera center C to g while keeping it approximately at the desired distance D from the structure along the way, we use a local path planner based on potential fields (Khatib, 1986; Choset *et al.*, 2005). A potential function encoding the structure as obstacles in the neighborhood of the camera, as well as the goal g, is sampled in the form of a cost map on local 2D grid of size $2\rho \times 2\rho$ centered on the camera's current position, see Fig. 4.5. Assumption 2 guarantees that all the occupied cells in this cost map denote the structure itself. For k occupied cells centered at $\{x_j\}_{j=1}^k$, the potential function $N(x)$ is defined as

$$N(x) = \alpha \|x - g\|^2 + \sum_{j=1}^{k} I_j(x) d_j(x), \tag{4.1}$$

$$\text{with } d_j(x) = \frac{1}{\beta\|x - x_j\|}; I_j(x) = \begin{cases} 1 & \text{if } \|x - x_j\| \leq D \\ 0 & \text{otherwise,} \end{cases}$$

for some scalar parameters α, β. Here d_j is the repulsion from the j^{th} occupied cell, and is limited by I_j to a neighborhood of radius D around the cell. A path for the camera is obtained by following the negative gradient of N, i.e., $\dot{x} = -\nabla N(x)$. Denoting $J_x = \{j : I_j(x) = 1\}$ the occupied cells in the D-neighborhood of x, we have

$$-\nabla N(x) = 2\alpha(g - x) + \sum_{i \in J_x} \frac{1}{\beta\|x - x_i\|^3}(x - x_i). \tag{4.2}$$

Let $Q = \{x : J_x \neq \emptyset\}$ denote the region that is at distance at most D from the structure. Assuming a small value of β, the summation term in (4.2) is dominant whenever $x \in Q$ and pushes the path away from the structure. However, this term vanishes as soon as $x \notin Q$. Then, assuming that the camera starts at x_0 on the boundary ∂Q of Q, it remains approximately on ∂Q if $g - x$ points toward the interior of Q. It is possible that this condition is not satisfied by the point g computed in the previous subsection, in which case we replace g by g_1, which is obtained by selecting a new $goal = \overline{p}^{\mathsf{C}} - D'\,\mathbf{n} + step\,\mathbf{r}$ for $D' < D$ such that this condition is satisfied. The path will then slide on ∂Q until it reaches its goal (Cortes, 2008). Finally, this path for the center C of the camera is used to compute a corresponding path for the center R of the robot that needs to be tracked using a platform specific controller.

4.3.3 Replanning due to the structure interferring

Assumption 2 guarantees that the robot can move sufficiently freely around the structure, but this does not prevent the structure itself from interfering with the path planned above. Consider the situation shown in Fig. 4.6a. The wall ahead of the robot does not fall into the FOV of the camera due to the limited horizontal angle of view, yet the robot should not approach this wall closer than a distance D. Hence, if the robot detects obstacles in its D-neighborhood, it is stopped at its current position and the yaw motion of the camera is used to scan ahead and face the new section of the structure. More precisely, as illustrated in Fig. 4.7, we use the costmap from the previous subsection to turn the camera to face along the direction from the robot center R to the first occupied

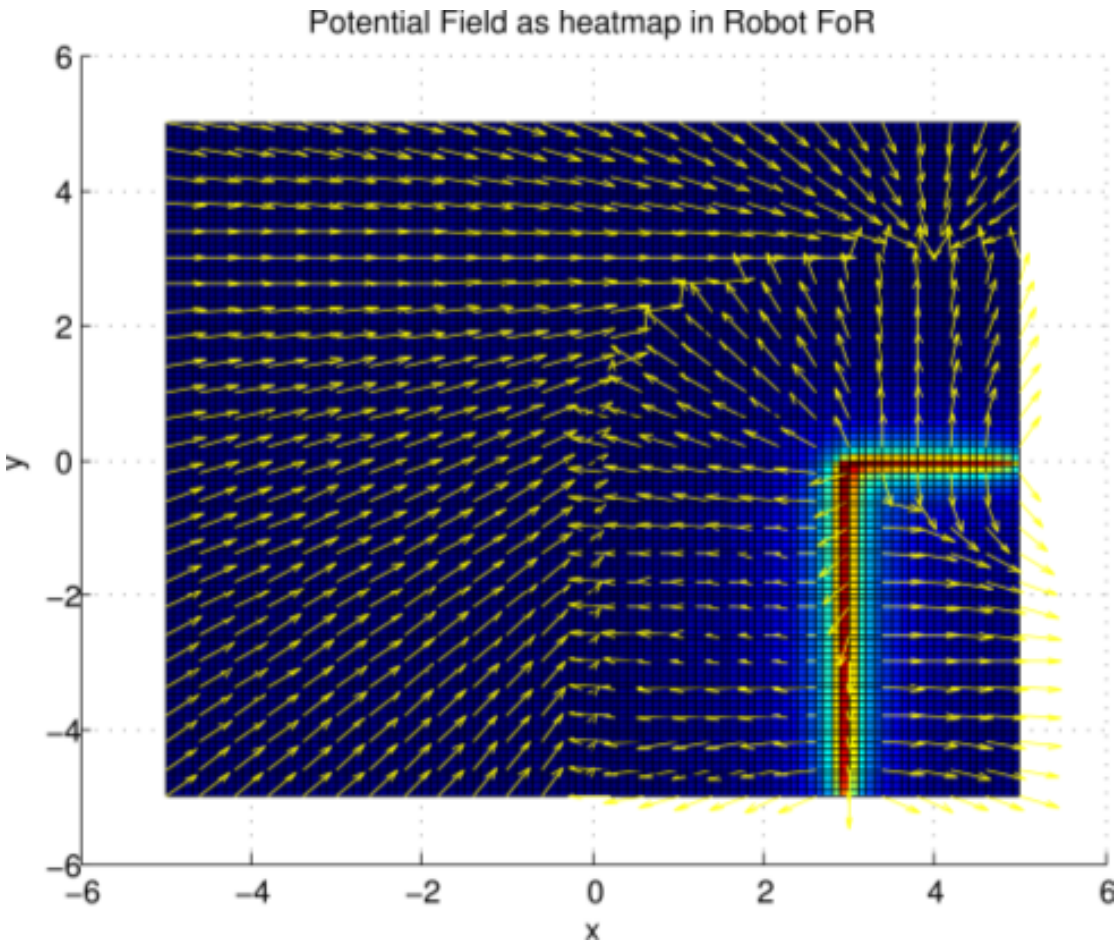
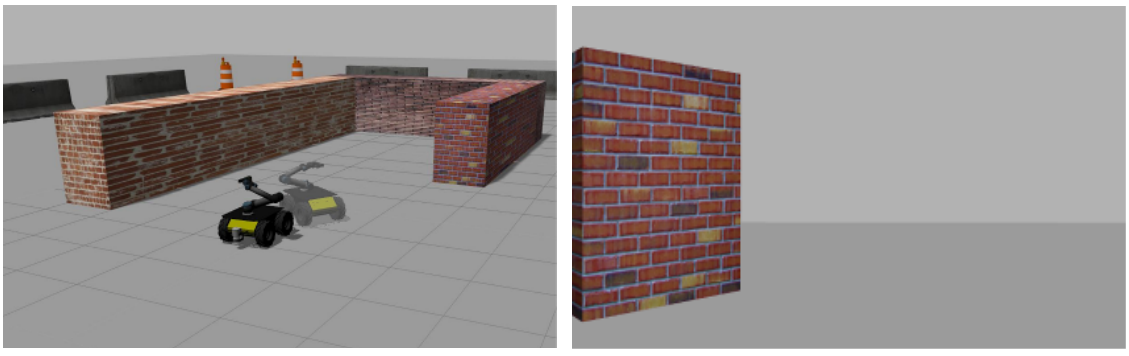


Figure 4.5: The potential field for a goal at $(4, 3)$ with $D = 3$ is shown as a heat map and the corresponding gradient vectors are shown as a vector field.



(a) (b)

Figure 4.6: a) While the robot is following the structure, its forward facing sensors detect an obstacle ahead (robot configuration shown in faded colors). This obstacle is outside the field of view of the camera, shown in b). The position of the robot at the next waypoint along the new direction to explore, determined by using the arm to scan ahead, is shown in bright colors.

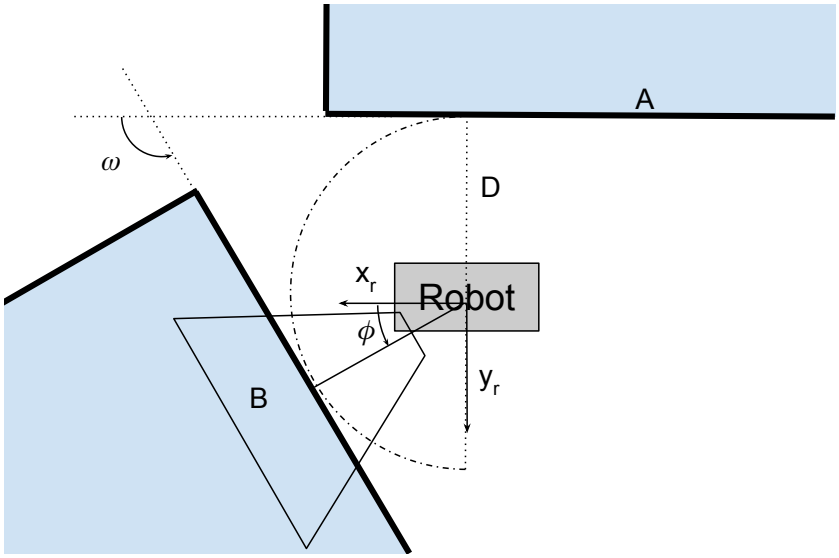


Figure 4.7: When the robot is currently following section A of the structure, a later section B of the structure could interfere with the planned path. In general, the angle ω made by section B with respect to section A satisfies $\omega \in [0, \pi)$. Also, the two sections could be connected to form a non-convex corner.

cell in the D-neighborhood of the robot. The next goal is then recomputed using the newly captured point cloud.

4.3.4 End of the PE phase

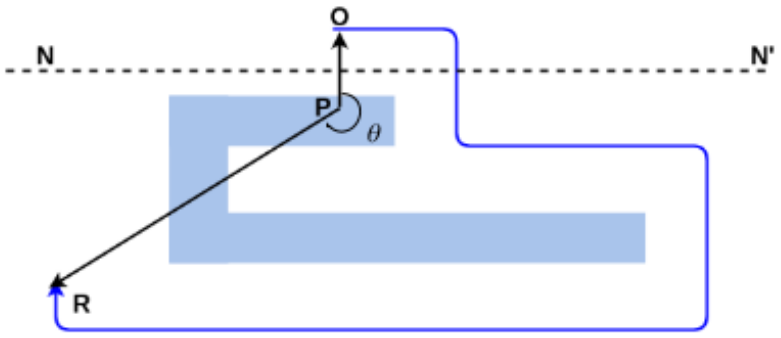


Figure 4.8: The angle θ subtended by the path $\widehat{\mathsf{OR}}$ at a reference point P inside the structure can be used to detect the end of the PE phase.

The end of the PE phase is determined by monitoring the angle θ subtended by the robot's trajectory with respect to a reference point P inside the structure, see Fig. 4.8. The point P is chosen from the first depth image at the start of the PE phase. As shown in Fig. 4.8, the ray PR completes one full rotation in the clockwise manner with respect to ray PO only at the end of the first pass. Note that θ need not monotonically increase and it depends on the geometry of the structure. But since by Initial Condition 1 there

exists a plane NN' that initially separates the robot and the structure, θ can complete a full rotation only at the end of the first pass. Additionally, after θ completes a full rotation, we continue to follow the structure until the vSLAM module achieves a global loop closure.

Overall, our strategy attempts to maintain as much as possible a viewpoint orthogonal to the structure, even though it replans for a new goal according to Algorithm 3 only at discrete times. Note that only the computation of the next goal happens at discrete instants but the vSLAM module updates the model at a higher rate as per the capabilities of the hardware.

4.4 Completing the Model

4.4.1 Determining flaws

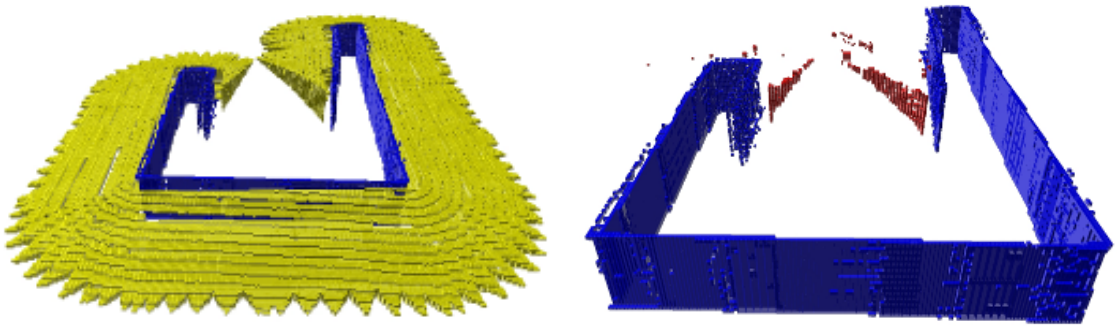
There are two possible types of flaws in the model obtained at the end of the PE phase. Type I flaws correspond to holes that are present in the already explored regions. As noted in Section II, these holes could be due to limitations of the sensor or local occlusions caused by small irregularities in the structure itself, and should be filled using a platform with a more appropriate reachable space, hence we do not consider them further. Type II flaws, called cavities in the following, correspond to regions that were missed during the PE phase, e.g., due to the situation depicted on Fig. 4.6, and will be filled during the CE phase. Our exploration policies only need the location of the entrances of the cavities and in this subsection we describe an algorithm to determine these locations in the model.

Our algorithm to determine the entrance to the cavities uses a voxel based 3D occupancy grid constructed from the global point cloud. We use the OctoMap (Hornung *et al.*, 2013) library to maintain this occupancy grid in a hierarchical tree data structure. Internally, the OctoMap library performs ray casting operations, labelling the occupancy measurement of each voxel along the line segment from the camera position to each point in the point cloud as *free* and the point itself as *occupied*. For this, we require the vSLAM module to provide the sequence of point clouds and associated camera positions used in assembling the current model. All voxels in the occupancy grid that are

not labeled free or occupied are called *unknown*.

Using the constructed OctoMap, we compute a set of frontier voxels, whose definition is adapted from (Yamauchi, 1997).

Definition 1 *A* frontier voxel *is a free voxel with at least one neighboring unknown voxel.*

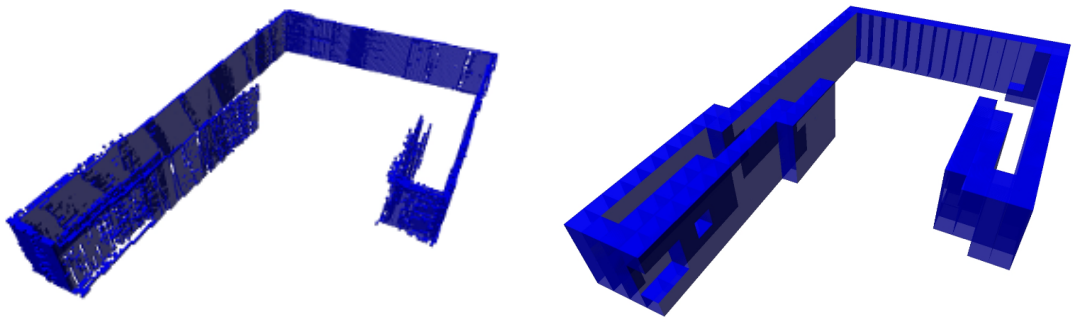


(a) with frontier voxels (b) with cavity entrance voxels

Figure 4.9: a) The constructed OctoMap with occupied voxels shown in blue and frontier voxels shown in yellow. These yellow voxels form the boundary of the explored region and most of them lie along the top and bottom faces of the view frustums. b) The cavity entrance voxels are shown in red.

Recall that the camera is constrained to move in a horizontal plane and the system continuously explored the structure in the clockwise sense during the PE phase. Consequently, point clouds captured before and after a cavity result in vertical planes of frontier voxels that border it and that we want to detect as entrance of the cavity for further exploration, see Fig. 4.9b. Therefore, we define *cavity entrance voxels* as frontier voxels satisfying two additional conditions. First, the normal to the frontier voxel, computed using the nearby frontier voxels(Mitra *et al.*, 2004), must approximately lie on the $\mathbf{x}_g\mathbf{y}_g$ plane. This serves to remove frontier voxels that lie along the top and bottom faces of the view frustums, See Fig. 4.9a. Second, Type I flaws can result in frontier voxels, which we want to exclude from cavity entrance voxels. Therefore, we require that the distance to the closest occupied voxel should be greater than some threshold d_0, which can be chosen as a small fraction of the distance maintained from the structure, say $0.1\,D$. As the number of voxels in a typical structure is very large, we do not perform this thresholding exactly but instead we use an estimate for the distance to the structure obtained from OctoMap. The hierarchical structure of OctoMap allows efficient multi-resolution queries, see Fig. 4.10, and thus we keep as cavity entrance voxels

only those that are marked free at a resolution of approximately d_0.



(a) Leaf size: $0.05m$, depth: 16 (b) Leaf size: $0.4m$, depth: 13

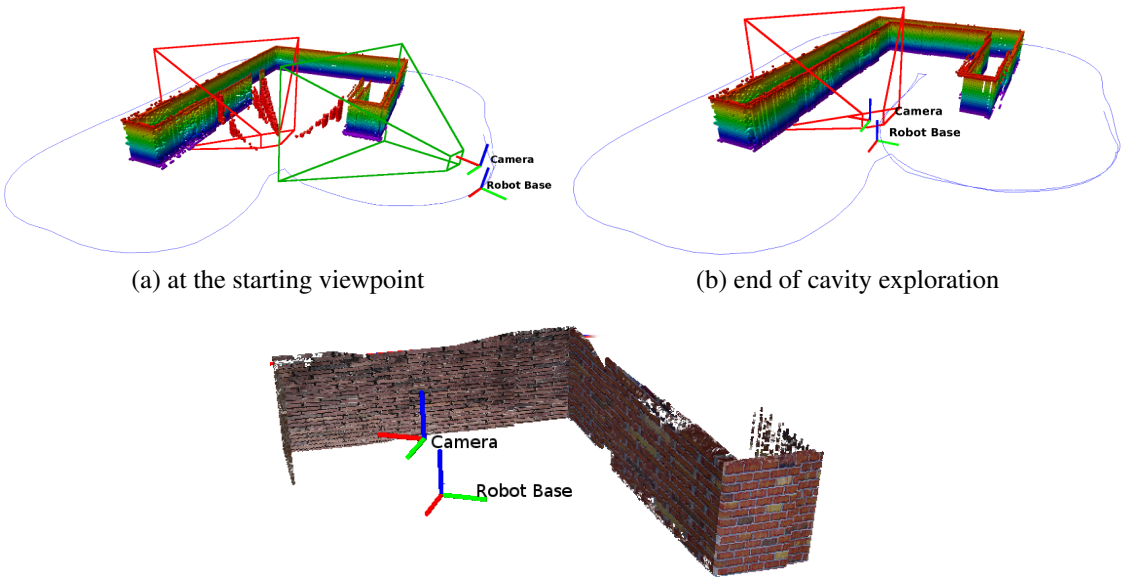
Figure 4.10: OctoMap queried at depth level 16 and 13 respectively. In this paper, the value of D is 3m and the threshold d_0 is chosen as 0.4m.

Finally, the cavity entrance voxels are clustered using an Euclidean clustering algorithm from PCL (Rusu and Cousins, 2011) and each cluster is referred to as a *cavity entrance*. Moreover, there could be some sparsely located cavity entrance voxels, which are removed by setting a minimum size for the cavity entrances.

4.4.2 Cavity Exploration

Once the cavity entrances have been determined, we can start the CE phase. We explore each detected cavity using an exploration strategy analogous to the PE phase. For this, we require a starting and ending viewpoint for each cavity entrance. We assume that there are no tunnels that go through the structure and that a robot entering a cavity can exit via the same location only. For a given cavity entrance, the starting viewpoint is chosen from the set of camera poses returned by the vSLAM module during the PE phase and such that the centroid of the cavity entrance lies within the view frustum. Additionally, the centroid should not be occluded by the structure from the camera position. From these camera poses, the one with the earliest timestamp τ_0 is chosen as starting viewpoint and we let τ_1 denote the largest timestamp. The ending viewpoint is chosen from the set of viewpoints from the PE phase as the one with the smallest timestamp τ_2 such that $\tau_2 > \tau_1$ and such that no cavity entrance voxel lies within the view frustum, see Fig. 4.11a.

The timestamps of the starting viewpoints of the cavity entrances are used to sort



(a) at the starting viewpoint

(b) end of cavity exploration

(c) mapping confined spaces

Figure 4.11: Cavity exploration. a) The robot is at the starting viewpoint for exploring the cavity. The starting and ending viewpoints for the first cavity entrance are shown in green and red respectively. All the cavity entrance voxels are also shown. b) The system detects the end of the cavity after coming close to the ending viewpoint. The blue line shows the trajectory followed by the robot. c) The extracted model showing CE in progress.

them in increasing order and each of the cavities is explored in sequence. A typical cavity has at least two cavity entrances bordering it and it is possible to have more cavity entrances if there is a row of pillars, for example. During the CE phase, if the centroid of a cavity entrance falls within the view frustum of the current camera position and is not occluded by the structure, we remove that cavity entrance from our list.

Exploring confined regions during the CE phase requires certain modifications to the PE policy. Recall that the system skipped the cavities during the PE phase as the robot came closer than a distance D from the structure. Therefore during the CE phase, only the region directly ahead of the robot and within a distance $\delta < D$ is checked for interference of the computed path with the structure. Moreover, our local path planner returns paths that maintain a distance δ from the structure, see Fig. 4.12. For this, using the notation of Sections 4.3.1 and 4.3.2, we modify $goal$ as $goal \leftarrow \overline{p}^{\mathsf{C}} - \delta\,\mathbf{n} + step\,\mathbf{r}$ and transform to the global FoR to obtain the new point g. The distance δ is chosen by starting from a small value and increasing it until we reach a local minimum of $N(g)$, where N is defined in (4.1).

The system detects that it has finished exploring the current cavity by monitoring

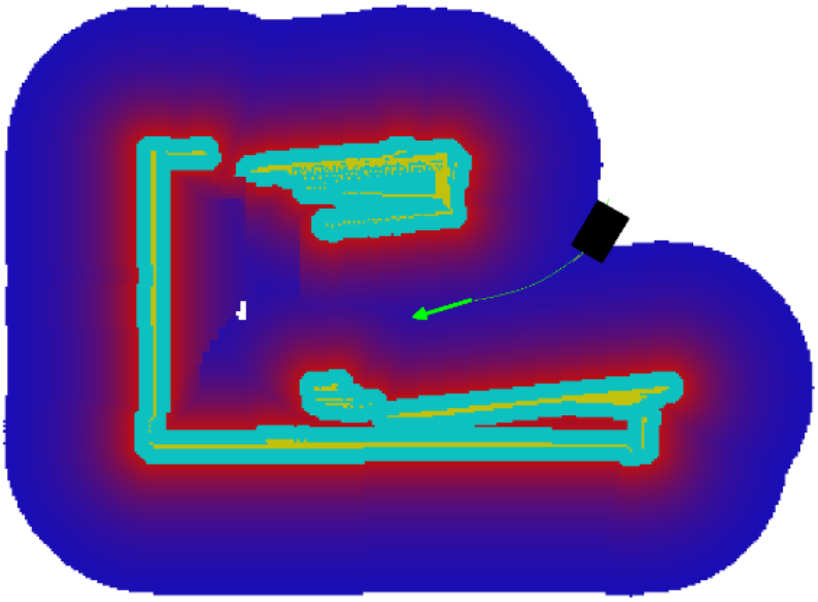


Figure 4.12: Costmap at the beginning of the exploration of a cavity. The robot is shown as a black rectangle and the green arrow oriented along $\mathbf{r}$ has its base at the computed next goal g such that $N(g)$ is a local minimum. The value of δ determined online is 2.0m while the value of D used in PE phase is 3.0m.

the loop closures obtained by the vSLAM module. When the robot exits the current cavity, the point clouds captured by the camera correspond to parts of the structure that are already present in the model, see Fig. 4.11b. Consequently, these result in loop closures in the vSLAM module. We declare the cavity explored once the system receives a loop closure with a viewpoint at timestamp τ such that $\tau_2 < \tau < \tau_F$ where τ_F corresponds to the last viewpoint of the PE phase. Alternatively, the number of changes in the occupancy measurements of the OctoMap could be used to detect the end of the cavity, since the point clouds captured after exiting the cavity do not add new information to the OctoMap. But this solution tends to be less robust because the localization errors and sensor noise can induce a large number of changes even when the camera is viewing a region that is already present in the model.

4.5 Simulations and Results

We evaluate the performance of our policies via 3D simulations for different sizes of the structure, camera range values and localization accuracy levels for the robot. The implementation of our motion planning policies is integrated with the Robot Operating System (ROS) Navigation Stack (Marder-Eppstein), which is supported by many

Table 4.1: Simulation results for different sizes of the structure and range of the camera

Model	Perimeter	Camera Range	Path Length
Small Γ	42m	4.5m	72.08m
Small Γ	42m	12.0m	53.79m
Large Γ	84m	4.5m	106.23m
Large a	94m	4.5m	179.65m

mobile ground robots. All the simulations are performed using the Gazebo simulator (Andrew Howard). The vSLAM algorithm used is RTAB-Map (Labbe and Michaud, 2014).

The simulations are carried out with publicly available models of a Clearpath Husky A200 robot and a Kinect depth sensor whose range can be varied (Hsu), see Fig. 4.2. A UR5 robotic arm is used to carry the sensor, but only yaw motions of the arm are allowed, as described in Section 4.2. For illustration purposes, we consider artificial structures made of short wall-like segments. We refer to the structure used in most of the previous illustrations as the Small Γ model. The Large Γ model has the same shape as Small Γ but is twice the size. We also illustrate the effectiveness of our policy for a realistic model of a house, and compare its performance with that of the classic Frontier-Based Exploration (FBE) algorithm (Yamauchi, 1997). We have included a supplementary MP4 format video, which shows the simulation of a Husky robot following our policies for mapping the Small Γ model using a Kinect sensor with a range of 4.5m.

4.5.1 Structure Size and Camera Range

The relative size of the structure with respect to the range of the camera affects the trajectory determined by our algorithms. Fig. 4.13 shows simulation results for 4 scenarios. With a camera range of 4.5m, the Large Γ model is completely mapped at the end of the PE phase. For the Small Γ model a cavity remains, which is subsequently explored during the CE phase. Increasing the camera range to say 12m allows the Small Γ structure to be mapped at the end of the PE phase as well, see Fig. 4.13b. Fig. 4.13d shows that the robot following our policies is able to map large structures with multiple cavities of different sizes. Table 4.1 lists the path lengths obtained for the different test

Table 4.2: Simulation results for different levels of localization accuracy

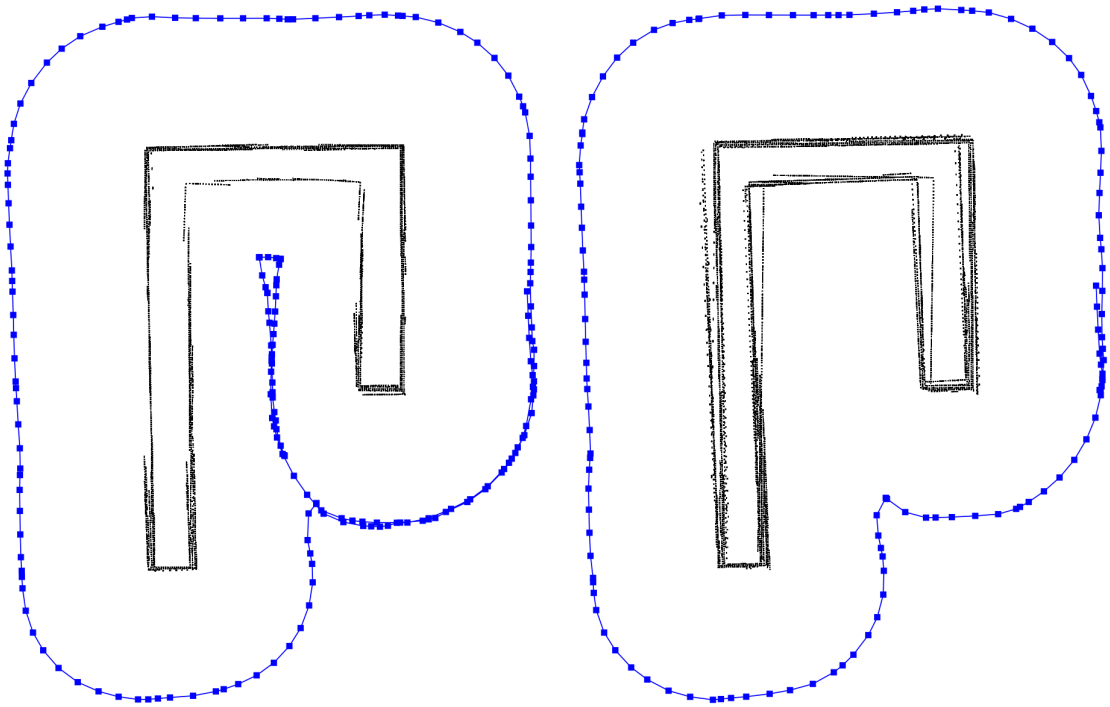
k	n_k	μ	σ	max
0.00	65520	0.05m	0.04m	0.20m
0.25	72636	0.08m	0.06m	0.27m
0.50	82728	0.11m	0.09m	0.40m
0.75	111321	0.16m	0.13m	0.94m

cases.

4.5.2 Localization accuracy

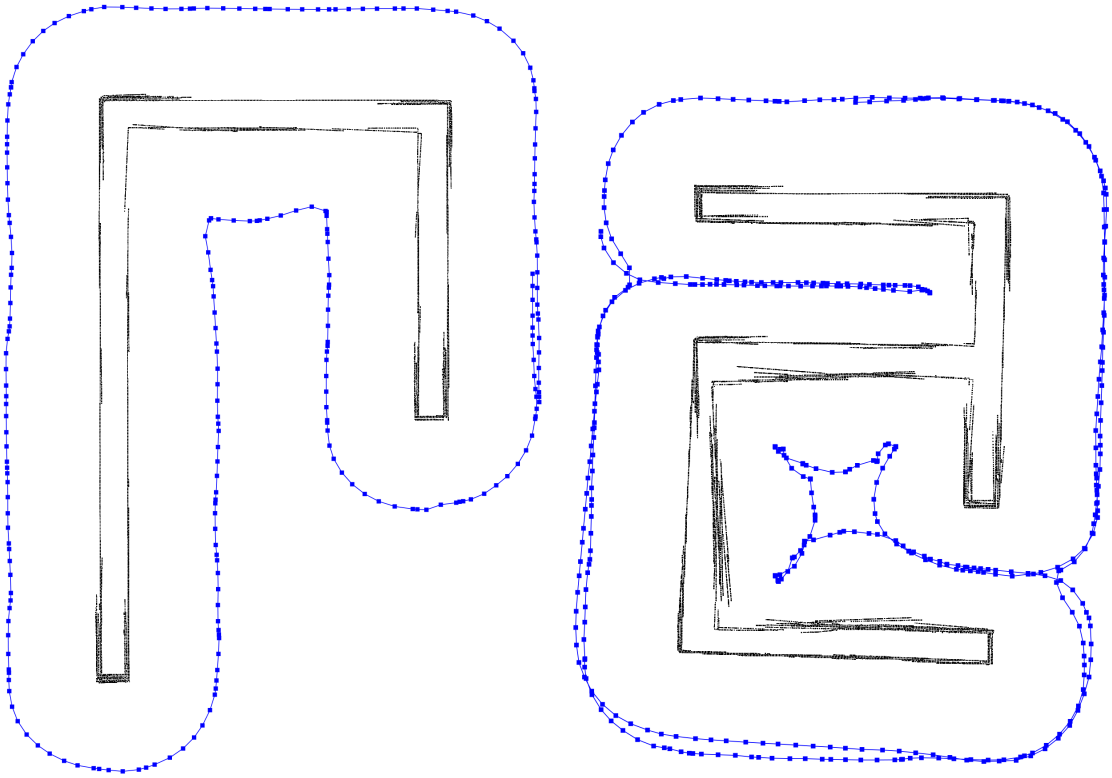
The Husky robot combines data from an Inertial Measurement Unit (IMU), a GPS module and wheel odometry to achieve a relatively small localization error overall. In order to evaluate the impact of localization accuracy on our algorithms, we simulate the effect of large wheel slippage by introducing a zero mean additive Gaussian white noise to each of the wheel encoder measurements, with a variance equal to $k(v_x + \omega_z)/2$, where v_x is the linear velocity of the robot, ω_z is its yaw rate and k is a proportionality constant, also called noise level in the following. Increasing k results in a poorer alignment of the point clouds, but all portions of the structure, except the horizontal faces, are still captured in the reconstructed model, see Fig. 4.14. Note that our policies compute the next waypoint at discrete times and therefore assume that the drift in localization between waypoints is sufficiently small so that the robot reaches the next waypoint with the camera facing the structure.

We use the CloudCompare(Girardeau-Montaut) software to compute the distortion in the reconstructed model $\mathcal{C}_k$, for a noise level k, with respect to a reference point cloud $\mathcal{C}_R$ which is generated using a different mobile platform with almost perfect localization. First, we register $\mathcal{C}_k$ to $\mathcal{C}_R$ using an Iterative Closest Point (ICP) algorithm (Rusinkiewicz and Levoy, 2001). We then define for every point in $\mathcal{C}_k$, its error to be the distance to the nearest neighbor in $\mathcal{C}_R$. Table 4.2 lists the simulation results for mapping the Small Γ model with different noise levels k, where n_k is the number of points in $\mathcal{C}_k$ and $\mu, \sigma, \max$ are respectively the mean, standard deviation and maximum value of the errors of all points in $\mathcal{C}_k$. The table indicates that both the mean and standard deviation of the errors increase with the noise level.



(a) Model: Small Γ, Range: $4.5m$

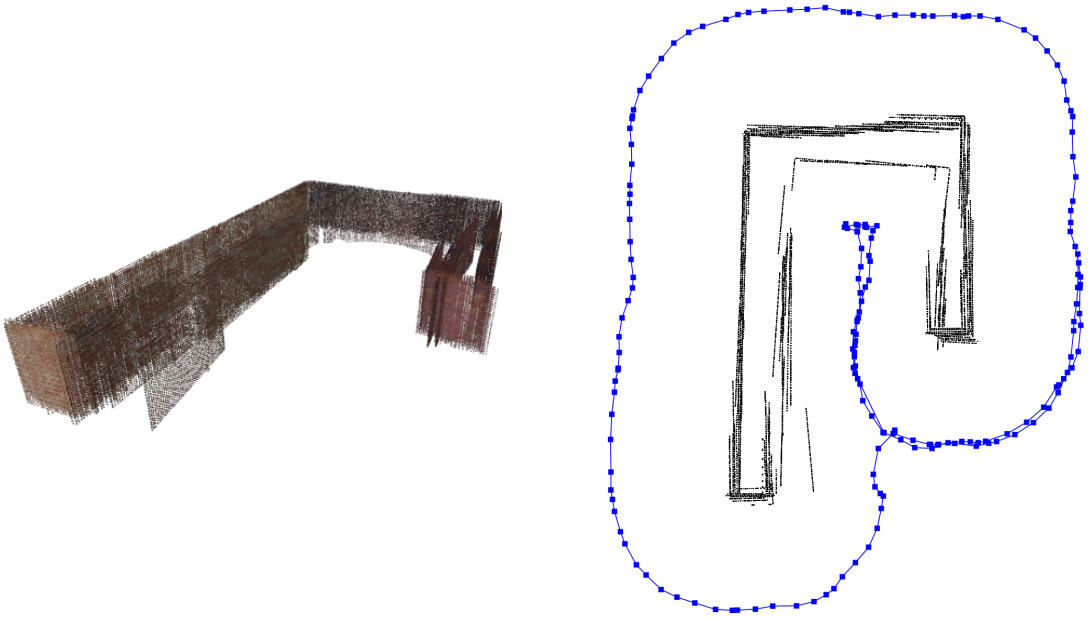
(b) Model: Small Γ, Range: $12m$



(c) Model: Large Γ, Range: $4.5m$

(d) Model: Large a, Range: $4.5m$

Figure 4.13: The projection of the reconstructed model on the $x_g y_g$ plane is shown in black and the trajectory followed by the robot based on our policies is shown in blue.



(a) Reconstructed model with $k = 0.75$

(b) Model: Small Γ, Range: 4.5m

Figure 4.14: a) Large errors in localization results in poor alignment, although all portions of the structure have been captured in the model, see Fig. 4.15a for comparison. b) The projection of the reconstructed model on the $\mathbf{x_g y_g}$ plane, when compared to Fig. 4.13a, shows the distortion introduced due to the noisy wheel odometry.

4.5.3 Comparing with Frontier-based Exploration

The Frontier Exploration(Bovbel) package available in ROS relies on a 2D LIDAR to build an occupancy grid that is used to compute the frontiers. The package requires the user to define a 2D polygon that encloses the structure. The algorithm then explores until there are no more frontiers inside the user-defined polygon. In comparison to the FBE algorithm, our algorithms

- do not require a user defined bounding polygon;
- maintain as much as possible a fixed distance from the structure (during the PE phase), thereby ensuring that all portions up to a height of $H_{\max}$ are mapped;
- consistently explore the structure in the clockwise direction, which can be important from a user perspective to understand the behavior of the robot. On the other hand, the trajectory prescribed by the FBE algorithm depends on the size of the user-defined bounding polygon. A large bounding polygon will cause the robot to explore areas far away from the structure and will possibly not maintain a fixed direction of exploration.
- produces the same path every time for a given structure whereas the path computed by the FBE algorithm could differ greatly between two trials. Also, the

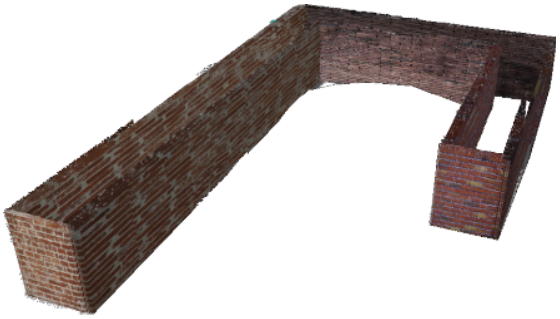
Table 4.3: Comparison between our policy and frontier based exploration

		Proposed Policy	FBE
Small Γ	Path Length	72.08m	49.78m
	Unique closest point set size	$6,063$	$5,398$
	Mean Error	0.05m	0.12m
House	Path Length	59.89m	47.55m
	Unique closest point set size	$9,182$	$7,402$
	Mean Error	0.05m	0.12m

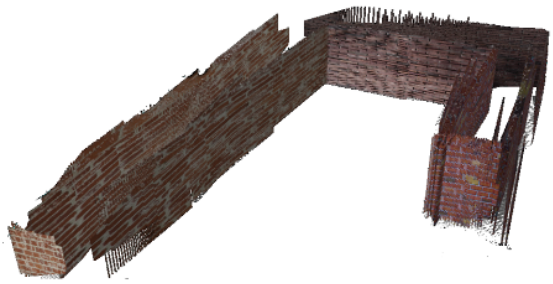
robot often gets stuck while using the FBE algorithm as the computed waypoints are often too close to the structure.

In order to get a quantitative measure of the structure coverage, we use again the CloudCompare software to compute essentially the projection of the reconstructed model $\mathcal{C}$ on the reference point cloud $\mathcal{C}_R$. Namely, for each point in $\mathcal{C}$ we compute the closest point in $\mathcal{C}_R$. Note that multiple points in $\mathcal{C}$ can have the same closest point in $\mathcal{C}_R$. In this case, we remove these duplicate points to obtain the *unique closest point set*. Then, as long as the reconstructed model aligns relatively well with the reference model, the cardinality of the unique closest point set is taken as our estimate of the structure coverage. Table 4.3 compares the level of structure coverage achieved by our policies and FBE with a camera range of 4.5m for two of the environments considered. For the Small Γ model, our reference point cloud has $6,116$ points with a minimum distance of 0.1m between points. For the House model, our reference point cloud has $10,889$ points with a minimum distance of 0.1m between points. Since the height of the House model is more than $H_{\max}$, we only take the portion of the reconstructed model up to the height $H_{\max}$ for computing the structure coverage and mean error for the two algorithms.

Table 4.3 shows that our policies achieve a higher level of structure coverage than FBE for the environments considered and our proposed coverage metric. Note also that the smooth trajectory prescribed by our policies is beneficial to the vSLAM module to achieve a better alignment and a lower value for the mean error in the reconstructed model, especially if the robot localization accuracy is poor. A visual inspection of Fig. 4.15a and Fig. 4.15b shows the improvement in performance of the vSLAM module when using our algorithms compared to FBE.

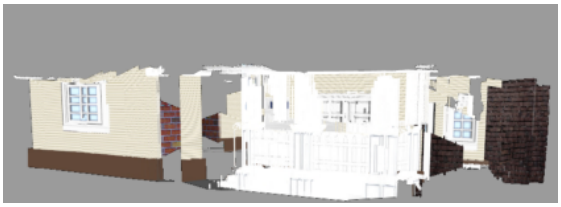


(a) Proposed Policy

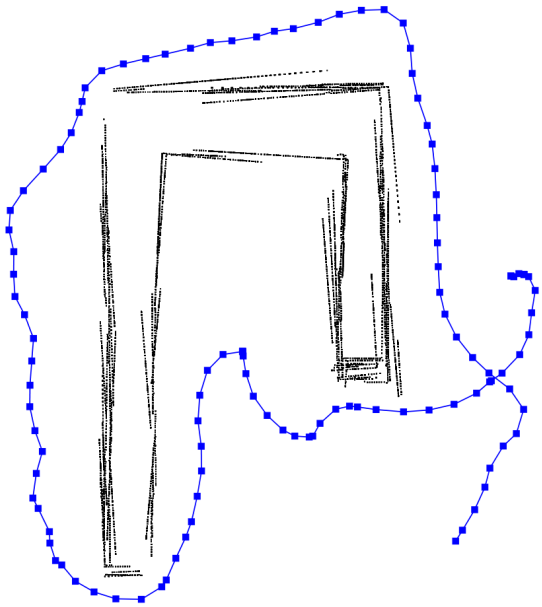


(b) Frontier based Exploration

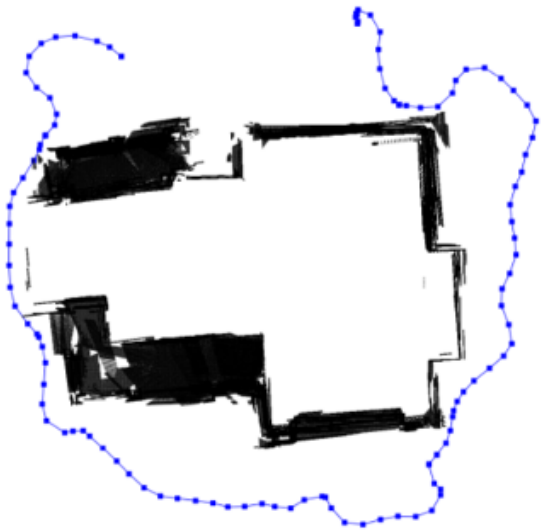
(c) Proposed Policy



(d) Frontier based Exploration



(e) Model: Small Γ, Range: $4.5m$



(f) Model: House, Range: $4.5m$

Figure 4.15: (a-d) Comparison of the reconstructed model using our policies and FBE. (e,f) The trajectory prescribed by FBE for the Small Γ and House structure is shown in blue.

4.6 Conclusion and Future Work

This work presents novel motion planning policies that guide a mobile ground robot carrying a depth sensor to autonomously explore the visible portion of a bounded three-dimensional structure. The proposed policies do not assume any prior information about the size or geometry of the structure. Coupled with state-of-art vSLAM systems, our strategies are able to achieve high coverage in the reconstructed model, given the physical limitations of the platform. We have illustrated the efficacy of our approach via 3D simulations for different structure sizes, camera range and localization accuracy. In addition, a comparison of our policies with the classic frontier based exploration algorithms clearly shows the improvement in performance for a realistic structure such as a house.

This work opens interesting questions such as combined vehicle and 6-DOF arm motion planning to map fine structural details. Extending our algorithms to aerial platforms and hybrid teams of robots, for example, would also allow for autonomous inspection of tall structures such as wind turbines or telecom towers.

REFERENCES

1. **Acar, E.**, **H. Choset**, and **J. Lee** (2006). Sensor-based coverage with extended range detectors. *IEEE Transactions on Robotics*, **22**(1), 189–198.

2. **Andrew Howard, N. K.** (). Gazebo robot simulator. URL `http://gazebosim.org/`. Accessed: 2015-12-29.

3. **Atanasov, N.**, **J. Le Ny**, **K. Daniilidis**, and **G. J. Pappas**, Decentralized active information acquisition: Theory and application to multi-robot SLAM. *In IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

4. **Bajcsy, R.** (1988). Active perception. *Proceedings of the IEEE*, **76**(8), 966–1005.

5. **Bircher, A.**, **K. Alexis**, **M. Burri**, **P. Oettershagen**, **S. Omari**, **T. Mantel**, and **R. Siegwart**, Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. *In IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, Washington, 2015.

6. **Bovbel, P.** (). ROS frontier exploration. URL `http://wiki.ros.org/frontier_exploration`. Accessed: 2015-11-26.

7. **Choset, H.**, **K. M. Lynch**, **S. Hutchinson**, **G. A. Kantor**, **W. Burgard**, **L. E. Kavraki**, and **S. Thrun**, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, 2005.

8. **Cortes, J.** (2008). Discontinuous dynamical systems. *IEEE Control Systems Magazine*, **28**(3), 36–73.

9. **Dornhege, C.** and **A. Kleiner** (2013). A frontier-void-based approach for autonomous exploration in 3D. *Advanced Robotics*, **27**, 459–468.

10. **El-Hakim, S. F.**, **J. A. Beraldin**, **M. Picard**, and **G. Godin** (2004). Detailed 3D reconstruction of large-scale heritage sites with integrated techniques. *IEEE Computer Graphics and Applications*, **24**(3), 21–29.

11. **Endres, F.**, **J. Hess**, **J. Sturm**, **D. Cremers**, and **W. Burgard** (2014). 3-D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, **30**(1), 177–187.

12. **Englot, B.** and **F. S. Hover**, Sampling-based coverage path planning for inspection of complex structures. *In International Conference on Automated Planning and Scheduling (ICAPS)*. Sau Paulo, Brazil, 2012.

13. **Frahm, J.-M.**, **P. Fite-Georgel**, **D. Gallup**, **T. Johnson**, **R. Raguram**, **C. Wu**, **Y.-H. Jen**, **E. Dunn**, **B. Clipp**, **S. Lazebnik**, and **M. Pollefeys**, Building Rome on a cloudless day. *In Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 3-642-15560-X, 978-3-642-15560-4.

14. **Furukawa, Y.** and **J. Ponce** (2010). Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **32**(8), 1362–1376.

15. **Girardeau-Montaut, D.** (). CloudCompare (version 2.6.0) [GPL software]. URL `http://www.cloudcompare.org`.

16. **Heng, L.**, **A. Gotovos**, **A. Krause**, and **M. Pollefeys**, Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. *In IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, Washington, 2015.

17. **Hornung, A.**, **K. M. Wurm**, **M. Bennewitz**, **C. Stachniss**, and **W. Burgard** (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, **34**(3), 189–206.

18. **Hsu, J.** (). Gazebo plugins. URL `http://wiki.ros.org/gazebo_plugins`. Accessed: 2015-12-22.

19. **Isaacs, R.**, *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1999.

20. **Jacobi, M.** (2015). Autonomous inspection of underwater structures. *Robotics and Autonomous Systems*, **67**(C), 80–86. ISSN 0921-8890. Special issue on Advances in Autonomous Underwater Robotics.

21. **Khatib, O.** (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, **5**(1), 90–98.

22. **Krainin, M.**, **B. Curless**, and **D. Fox**, Autonomous generation of complete 3D object models using next best view manipulation planning. *In IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

23. **Kriegel, S.**, **C. Rink**, **T. Bodenmüller**, and **M. Suppa** (2015). Efficient next-best-scan planning for autonomous 3D surface reconstruction of unknown objects. *Journal of Real-Time Image Processing*, 611–631.

24. **Kuemmerle, R.**, **G. Grisetti**, **H. Strasdat**, **K. Konolige**, and **W. Burgard**, g2o: A general framework for graph optimization. *In IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

25. **Labbe, M.** and **F. Michaud**, Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Chicago, IL, 2014.

26. **Lau, G.** and **H. H. Liu** (2014). Real-time path planning algorithm for autonomous border patrol: Design, simulation, and experimentation. *Journal of Intelligent and Robotic Systems*, **75**(3-4), 517–539.

27. **Lim, R.**, **H. M. La**, and **W. Sheng** (2014). A robotic crack inspection and mapping system for bridge deck maintenance. *IEEE Transactions on Automation Science and Engineering*, **11**(2), 367–378.

28. **Lin, J.**, **K. Han**, and **M. Golparvar-Fard**, A framework for model-driven acquisition and analytics of visual data using UAVs for automated construction progress monitoring. *In Computing in Civil Engineering*. 2015.

29. **Marder-Eppstein, E.** (). ROS navigation stack. URL `http://wiki.ros.org/navigation`. Accessed: 2015-11-26.

30. **Mitra, N. J.**, **A. Nguyen**, and **L. Guibas** (2004). Estimating surface normals in noisy point cloud data. *International Journal of Computational Geometry & Applications*, **14**(04n05), 261–276.

31. **Rusinkiewicz, S.** and **M. Levoy**, Efficient variants of the ICP algorithm. *In Proceedings of the Third International Conference on 3-D Digital Imaging and Modeling, 2001*. Quebec City, Canada, 2001.

32. **Rusu, R. B.** and **S. Cousins**, 3D is here: Point cloud library (PCL). *In IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

33. **Shade, R.** and **P. Newman**, Choosing where to go: Complete 3D exploration with stereo. *In IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

34. **Shen, S.**, **N. Michael**, and **V. Kumar**, Autonomous indoor 3D exploration with a micro-aerial vehicle. *In IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, Washington, 2012.

35. **Sheng, W.**, **H. Chen**, and **N. Xi** (2008). Navigating a miniature crawler robot for engineered structure inspection. *IEEE Transactions on Automation Science and Engineering*, **5**(2), 368–373.

36. **Venkatesan, R. H.** and **N. K. Sinha**, The target guarding problem revisited: Some interesting revelations. *In Proceedings of the 19th IFAC World Congress*. International Federation of Automatic Control, 2014.

37. **Wu, C.** (2011). VisualSFM: A visual structure from motion system. `http://ccwu.me/vsfm/`. Accessed: 2015-12-27.

38. **Yamauchi, B.**, A frontier-based approach for autonomous exploration. *In Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, CIRA '97. IEEE Computer Society, Washington, DC, USA, 1997. ISBN 0-8186-8138-1.

39. **Zhou, S.**, **J. Xi**, **M. W. McDaniel**, **T. Nishihata**, **P. Salesses**, and **K. Iagnemma** (2012). Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain. *Journal of Field Robotics*, **29**(2), 277–297.

LIST OF PAPERS BASED ON THESIS

1. Manikandasriram Srinivasan Ramangopal, Jerome Le Ny Motion Planning Strategies for Autonomously Mapping Unknown 3D Structures *IEEE Transactions on Automation Science and Engineering* (under review)