Michigan Technological University
Digital Commons @ Michigan Tech

Dissertations, Master's Theses and Master's Reports - Open

Dissertations, Master's Theses and Master's Reports

2014

AUTONOMOUS POWER DISTRIBUTION SYSTEMS

Barzin Moridian
Michigan Technological University

Follow this and additional works at: https://digitalcommons.mtu.edu/etds

Part of the Robotics Commons

Copyright 2014 Barzin Moridian

Recommended Citation

Moridian, Barzin, "AUTONOMOUS POWER DISTRIBUTION SYSTEMS", Master's Thesis, Michigan Technological University, 2014.
https://doi.org/10.37099/mtu.dc.etds/875

Follow this and additional works at: https://digitalcommons.mtu.edu/etds
Part of the Robotics Commons

AUTONOMOUS POWER DISTRIBUTION SYSTEMS

By

Barzin Moridian

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Mechanical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2014

© 2014 Barzin Moridian

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Mechanical Engineering.

Department of Mechanical Engineering - Engineering Mechanics

Thesis Advisor: *Dr. Nina Mahmoudian*

Committee Member: *Dr. Wayne W. Weaver*

Committee Member: *Dr. Rush Robinett*

Department Chair: *Dr. William W. Predebon*

Contents

List of Figures . . . . **ix**

Dedication . . . . **xiii**

Abstract . . . . **xv**

1 Introduction **1**

1.1 Motivation 1

1.2 Scope and Objectives 4

1.3 Contribution and Outline 6

2 Path Planner **9**

2.1 Introduction 9

2.2 Literature Review 11

2.2.1 Cell Decomposition 11

2.2.2 Bug Algorithm 12

2.2.3 Vector Field Histogram 14

2.2.4 Dynamic Window Approach 17

2.2.5 Elastic Band 18

2.2.6 Potential Field 20

2.2.7 Fractional Potential Field Method 22

2.3 Robust Path Planner 23

2.3.1 Modeling 24

2.3.2 Algorithm 28

2.4 Robust Path Planning Algorithm Results 32

3 Navigation for Power Distribution Systems 35

3.1 Multi-layered Control Hierarchy Structure 36

3.2 Mission Planner 38

3.3 Simple Path Planner 40

3.4 Path tracking 43

4 Power Distribution Experimental Results 47

4.1 Scenario 47

4.2 Hardware 51

4.2.1 Power Electronics 51

4.2.2 Ground Robot 56

4.2.3 Magnetic Electrical Connections 56

4.2.4 Camera System and IR Sensor 58

4.3 Results 60

4.3.1 IR Filtering and Direction Choosing Algorithms Results for Simple Path Planning Algorithm 60

4.3.2 Simple Path Planning Algorithm Results 62

5 Conclusion and Future Works 65

5.1 Conclusion 65

5.2 Future Developments 67

References 71

A Navigation Code for Unsophisticated Platform 85

B Robust Path Planning Algorithm Codes 87

B.1 Front End 87

B.2 Workspace Object Definition 90

B.3 Robot Object Definition 92

B.4 LIDAR Object Definition 122

C Letters of Permission 127

List of Figures

1.1 A team of autonomous microgrid robots re-establish power to a communication tower. Right: connection and generation agent, middle: two renewable energy agents, back left: power conversion agent 3

2.1 Bug 1 algorithm performance 13

2.2 Bug 2 algorithm performance 14

2.3 Example of a setting where Bug 1 outperforms Bug 2 15

2.4 Environment ©1991 IEEE 16

2.5 Cartesian Histogram Model ©1991 IEEE 16

2.6 Polar density in Cartesian and polar form before applying threshold ©1991 IEEE 17

2.7 An example setting for dynamic window approach ©1997 IEEE 18

2.8 Admissible velocities ©1997 IEEE 18

2.9 Dynamic window representation ©1997 IEEE 19

2.10 ElasticBand ©1993 IEEE 20

2.11 Unstable performance of potential field in narrow corridors ©1991 IEEE . 21

2.12 Environment representation 25

2.13 Two dimensional sensor representation 26

2.14 Path representation 27

2.15 Obstacle Segmentation 30

2.16 subgoal candidates 30

2.17 Subgoals candidates and the best subgoal: Best candidate has a purple highlight, final destination is the unfilled blue circle on the right side 31

2.18 Test results of robust path planning algorithm in an environment similar to indoors 32

2.19 Test results of robust path planning algorithm in an environment similar to outdoors 33

3.1 Control hierarchy of 2 agents 37

3.2 Path planning and target reaching algorithm flowchart 41

3.3 Direction choosing algorithm using IR scan 42

4.1 Initial state of power source robot, cabling robot, and two power loads . . . 49

4.2 power source robot moves towards one of the power loads (blue) and makes an electrical connection 50

4.3 Cabling robot moves towards the other power load (red) and makes an electrical connection 50

4.4 Cabling robot moves towards the power source robot and makes the final connection 50

4.5 A Microgrid Agent and a Power Load; Photo by author 51

4.6 Power Electronic Building Block (PEBB) 52

4.7 Types of conversion, dc-to-dc, dc-to-ac, and ac-to-ac through back-to-back dc to ac converters. 53

4.8 Photovoltaic panel and a regulated dc to dc converter with maximum power point tracking. 53

4.9 dc to dc conversion module 54

4.10 dc to ac conversion module 54

4.11 Combination of a power source robot and a cabling/conversion robot to power up a 48 V_{dc} and a 24 V_{dc} load 55

4.12 Male magnetic connector with wire pulley; Photo by author 57

4.13 Female magnetic connector; Photo by author 58

4.14 Marker placement on robot and objec; Photo by author 59

4.15 3D representation of objects in workspace by camera system software utilizing markers 60

4.16 Example of IR readings, scoring, and direction choosing results 61

4.17 Test results of path planning of mission of two agents providing power to two power loads 63

5.1 Four robot system. Robot A converts power from 24 V_{dc} source to 48 V_{dc} common bus. Robot B converts power from bus to 48 V_{dc} load. Robot C converts power to/from 12 V_{dc} battery. Robot D converts power to/from bus to 120 V_{ac} source. 67

5.2 Four robot system. Robot A converts power from 24 V_{dc} source to 48 V_{dc} common bus. Robot B converts power from bus to 48 V_{dc} load. Robot C converts power to/from 12 V_{dc} battery. Robot D converts power to/from bus to 120 V_{ac} source. Robot E provides renewable energy from a on-board solar panel and regulated dc to dc converter. 68

C.1 Figure permission from Wayne W. Weaver 128

C.2 Figure permission from Wayne W. Weaver; Continued 129

C.3 IEEE figures permission 130

Dedication

To my mother, teachers and friends

who did not hesitate to criticize my work at every stage - without which I would neither be who I am nor would this work be what it is today.

Abstract

Using robotic systems for many missions that require power distribution can decrease the need for human intervention in such missions significantly. For accomplishing this capability a robotic system capable of autonomous navigation, power systems adaptation, and establishing physical connection needs to be developed. This thesis presents developed path planning and navigation algorithms for an autonomous ground power distribution system. In this work, a survey on existing path planning methods along with two developed algorithms by author is presented. One of these algorithms is a simple path planner suitable for implementation on lab-size platforms. A navigation hierarchy is developed for experimental validation of the path planner and proof of concept for autonomous ground power distribution system in lab environment. The second algorithm is a robust path planner developed for real-size implementation based on lessons learned from lab-size experiments. The simulation results illustrates that the algorithm is efficient and reliable in unknown environments. Future plans for developing intelligent power electronics and integrating them with robotic systems is presented. The ultimate goal is to create a power distribution system capable of regulating power flow at a desired voltage and frequency adaptable to load demands.

Chapter 1

Introduction

1.1 Motivation

As our world advances technologically, we are continuing to find a plethora of new applications for the devices we utilize. One such example is the use of autonomous robots with new applications introduced everyday, in variety of fields. Examples of current applications include disaster relief/recovery efforts at Fukushima meltdown [1] and the prolonged search and rescue mission for the Malaysian MH370 flight [2]. However, as research is increasing, our scope of autonomous vehicles can also widen to include space, littoral, air, and ground efforts [3, 4, 5, 6]. Expanding ground efforts leads to many more possibilities. Some of the possible applications can include system monitoring in remote areas, power restoration, or even assistance to unintelligent systems [7].

Due to the increased use of autonomous devices and their applications, limitations are beginning to show in overall systems. One major issue becoming more evident is the lifetime of a device due to its power supply [8]. Having devices that are severely limited by their supply of power has researchers improve the strategic approach to an application. However, approaching this problem by conservative means such as optimizing applications, can limit the effort in optimizing the devices used [9]. In the long term this limited scope will lead to a fleet of devices that are confined to very specific tasks, wasting vital resources. Our contribution is to combine autonomous vehicles and algorithms capable of navigating complex and remote environments, supplying power to necessary devices.

To widen the scope of these autonomous devices, researchers have been developing ideas to provide systems with energy autonomy. Energy autonomy refers to a system's devices being intelligent and self-sufficient, recharging themselves when needed [10]. Energy autonomy would give a system and its devices a greatly increased time of operation instead of relying on equipped power sources [9]. Current research of energy autonomy is focused on providing power to a system through either another robot or a docking system via trophallaxis (donation through host power supply.) Either of these methods requires the robot to cease operation and make a physical connection to recharge [11, 12, 10], or performing a battery exchange [9, 13, 14].

An example system that may need emergency response in remote locations is a cellular telephone communication tower during a natural disaster [7].The conventional process in such instances was deploying personnel to power up the system using generators. This type of system is typically served by 208 V_{ac} three-phase power diesel backup generators. In such scenarios, energy autonomous systems can compensate for lack of human resources for power recovery efforts. This approach is not limited to ground vehicles and can be extended to underwater systems [3] or air vehicles [5, 6].

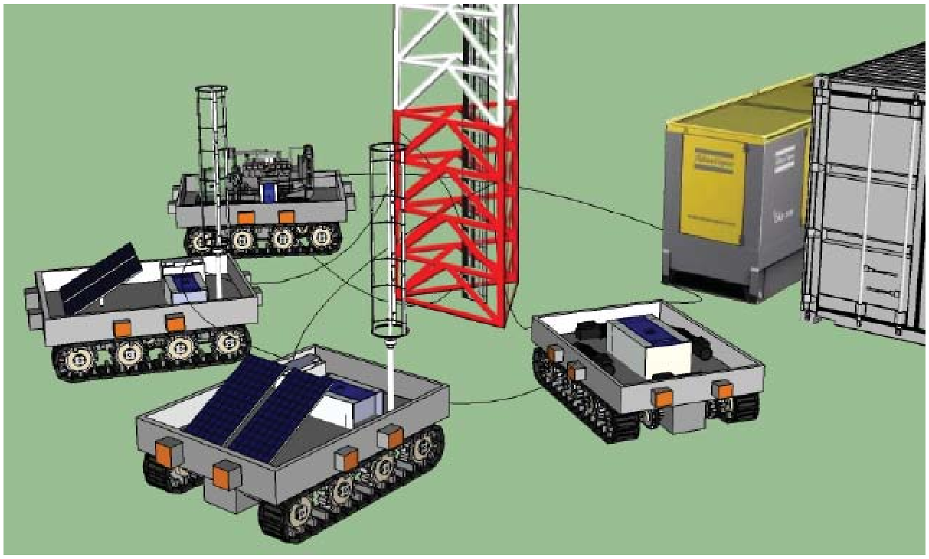


Figure 1.1: A team of autonomous microgrid robots re-establish power to a communication tower. Right: connection and generation agent, middle: two renewable energy agents, back left: power conversion agent

Addressing this example, Fig. 1.1 shows four autonomous microgrid robots with different power network functionality. Two of these four robots have renewable energy generation and storage capability, another has a conventional diesel genset, and the fourth contains intelligent power electronics for conversion and hard-line interconnection, and switchgear. After assessing the power requirements and available resources, the robots would physically organize and electrically interconnect to form a microgrid.

1.2 Scope and Objectives

The goal of this project is to create microgrids that are capable of self-configuration. A microgrid is a collection of energy resources that contribute to a common network. The resources include, but are not limited to, generation, conversion, loads, and storage [15]. Currently the power generation approach consists of two conventional models, centralized and distributed generation. Of these methods, centralized generation is being gradually replaced by distributed generation [16]. The model of distributed generation grants each individual device the ability to act as a power node if required by the system. Having this ability directly attests to the substantial flexibility this system has.

The idea of having an autonomous mobile power-grid is a recent application of autonomous vehicles and is currently in its initial stages of research [17]. In an autonomous power distribution system, each robot can act as a power node agent to help establish a microgrid. The system would be able to operate autonomously during disturbances in the utility network and increase reliability with minimal human interference [18]. In addition, the interconnection and networking of groups of microgrids can reduce the energy storage requirements to many differing applications.

Our plan is to find and validate practical solutions for deploying and controlling mobile microgrid systems and their capability to restore power quickly to various locations and equipment.

Deploying an autonomous microgrid system to re-establish electrical power for the crucial components of a system with minimal human interference can have a significant impact in many situations. This system can be more effective, with more consistency in operation, and enable power distribution faster than traditional methods and reduce the fuel delivery demands, if it:

† contains sources to generate electrical power

† has a self-configurable framework to reach optimal formation based on power demand

Our final objective is to integrate vehicle robotics with intelligent power electronics and electric power assets to create self-organizing, ad-hoc electric microgrids. These components ensure the capability to restore power quickly to critical need sites. We start with finding and validating a practical solutions for employing and control of a mobile powergrid.

1.3 Contribution and Outline

To best of our knowledge, we are among the first to integrate vehicle robotics with intelligent power electronics to create self-organizing, ad-hoc, microgrids. Our aim is to develop a control algorithm that accounts for uncertainty in predictions, requires minimal communication between agents, provides real-time guarantees on the performance of path planning, and reaches the targets while making electrical connections.

Some of the main challenges that can affect the practicality of such algorithms are that they have to guarantee 1) navigation of remote unknown environments and 2) complex design of power connectors.

For this project two path planning algorithms were developed. The first algorithm is a robust sensor-based, geometric, local path planning algorithm with the assumption of having a completely unknown environment. In this method the geometric properties of the robot are fully taken into consideration to guarantee reliable obstacle avoidance. This algorithm is capable of taking the sensor's limits into consideration and can be easily implemented on an embedded system. The second algorithm is a simple path planning algorithm that was developed with the idea of implementation on a lab-sized platform with limited functionality potential. The aim of this implementation was proofing the feasibility of using autonomous systems for power distribution.

Concepts and components used in this work are discussed as follows; Chapter 2 describes the developed path planning algorithm, it also includes a literature review of available local and global path planning methods. Navigation methods and a control hierarchy developed for autonomous power distribution proof-of-concept is discussed in chapter 3. A description of supporting hardware, validation of autonomous power distribution concept, and practicality of the path planning algorithms are studied in chapter 4. Conclusions derived from this study and future works are discussed in chapter 5.

Chapter 2

Path Planner

2.1 Introduction

Any autonomous system with moving parts needs a method for administering the trajectories. The reliability of an autonomous system, especially during its interaction with people, objects, and devices, is strongly related to dynamic properties of the robot and assumptions, considerations, and robustness of its dynamic control method. The navigation of robots (or more specifically, path planning of robots) is an old problem and has continuously improved. A proper navigation method should guarantee a safe and collision-free path and also it should try to optimize other desirable parameters such as vehicle's time of travel, time of path generation, robustness, flexibility, length of trajectory, compatibility with robot's maneuverability, etc.

Methods of solving trajectory planning problems can be put into two main categories, one is global path planning and the other is local path planning.

In global path planning methods, the problem is solved with the assumption of having a thorough knowledge about the environment in advance. Hence, the characteristics of sensors and their limitations usually do not play a big part in these methods. These type of methods try to solve the problem for optimality of different parameters, for example the shortest path is one of most popular parameters in previous studies. Although these methods can result in optimal solutions, they have limitations for implementation in applications where the environment is unpredictable or some information is missing.

On the other hand, in local path planning approaches, the robot learns about its environment while trying to reach its destination. Hence, output of these methods is constrained by the limitations of the robot's knowledge of its environment. These limitations include sensor's maximum distance range, sensor's angular range, number of measuring dimensions, position and geometric characteristics of the obstacle, dynamic characteristics of the obstacle, feasibility of generating a complete path, etc.

In this chapter, section 2.2 provides a survey on different path planning algorithms. Section 2.3 presents a path planning algorithm developed for robust obstacle avoidance in cluttered area. Finally, in section 2.4 results of this developed method are discussed.

2.2 Literature Review

In this section a review on different path planning methods is presented to show a history of limitations, assumptions, and achievements.

2.2.1 Cell Decomposition

One of the global path planning methods is cell decomposition in which all the free space in the environment is divided into non-overlapping cells. An adjacency relation of all the cells in the environment is then calculated. For calculating a successful path, a series of cells needs to be extracted. This series starts with the cells that include the start point and destination. From adjacency relation of the cells in an environment, a set of cells that connect these two cells is also added to the series.

There exist two main strategies for decomposing the environment into cells. One is exact cell decomposition [19] and the other is approximate decomposition [20, 21, 22]. The exact method can lead to difficulties in computation in case of having obstacles with irregular boundaries. In the approximate method, the free space is approximately covered by cells, making it easier to compute but it might result in ignoring some of the possible solutions.

2.2.2 Bug Algorithm

The original bug algorithm [23, 24, 25] introduce a minimal sensing method to help a robot to reach its target. Bug Algorithm does not need global knowledge about its environment. A global goal and local knowledge about the environment is enough for finding a complete path. This method can't handle dynamic environments though. In such environments, the position and geometric characteristics of obstacles are unknown and only the coordinates of the robot and target are known. Information about the surroundings essentially comes from tactile sensors but can be extended to proximity sensors. This information needs to be updated repeatedly during the mission. This method does not need to do an approximation of geometry of obstacles, but it does not guarantee an optimal path and the real-time motion planning. For this method, the ultimate performance expected from path generation is equal to the distance between the start and destination points and the perimeters of the obstacles that intersect with the line between these two points.

The conventional method that was introduced in [24] proposes two versions of Bug algorithms; Bug 1 and Bug 2. Each one of these algorithms have their relative benefits and disadvantages.

Bug 1 algorithm (example in Fig. 2.1) repeats the following steps until it reaches the target:

† robot starts moving towards the target along a straight line (M-line) until it reaches

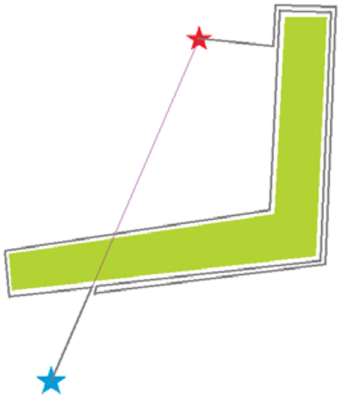


Figure 2.1: Bug 1 algorithm performance

the target or an obstacle.

- † if it reaches the target the algorithm terminates.
- † if it reaches an obstacle, the point that the obstacle was introduced to the robot is considered as a “hit point”. Then the robot should follow the obstacle boundary and measuring its distance from the target until it reaches the hit point again. The point with minimum distance from the target is set as the leave point which the robot moves towards it.

Bug 2 algorithm (example in Fig. 2.2) repeats the following steps until it reaches the target:

- † Robot starts moving towards the target along a straight line (M-line) until it reaches the target or an obstacle.
- † If it reaches the target, the algorithm terminates.
- † If it reaches an obstacle, the robot starts following the obstacle’s boundary until it

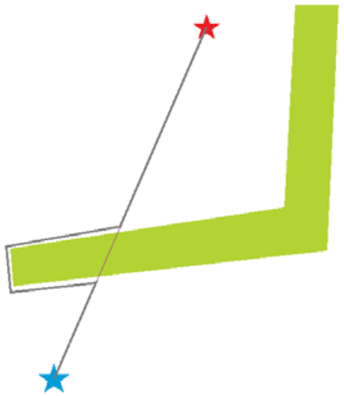


Figure 2.2: Bug 2 algorithm performance

reaches the straight line again at a point with less distance from the target than the hit point.

Bug 1 and Bug 2 outperform each other in different cases. Bug 1 in an exhaustive search algorithm but Bug 2 is a greedy algorithm. In Fig. 2.2 the Bug 2 algorithm outperforms Bug 1 algorithm for the example presented.

In contrast to Fig. 2.2, in some environments such as the one illustrated in Fig.2.3 Bug 1 performs better than Bug 2.

2.2.3 Vector Field Histogram

This method is developed with the idea of having an algorithm capable of fast detection and avoidance of unknown obstacles while moving towards the target [26, 27, 28]. This method is a local path planning algorithm and does not guarantee a complete or optimal path. In this

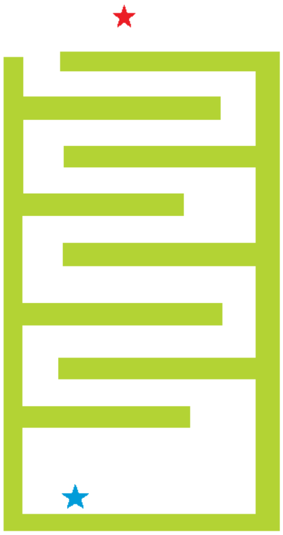


Figure 2.3: Example of a setting where Bug 1 outperforms Bug 2

method, the world surrounding the robot (Fig. 2.4) is modeled using a Cartesian histogram Fig. (2.5).

This histogram grid gets updated continuously via proximity sensors installed on the robot to equip the robot with flexibility of managing environment changes. This world model is then transformed from a two dimensional to a one dimensional polar histogram around the robot's position (Fig. 2.6). The next step is analyzing this polar histogram to find the direction with the lowest obstacle density. The chosen direction is set as the moving direction of the robot.

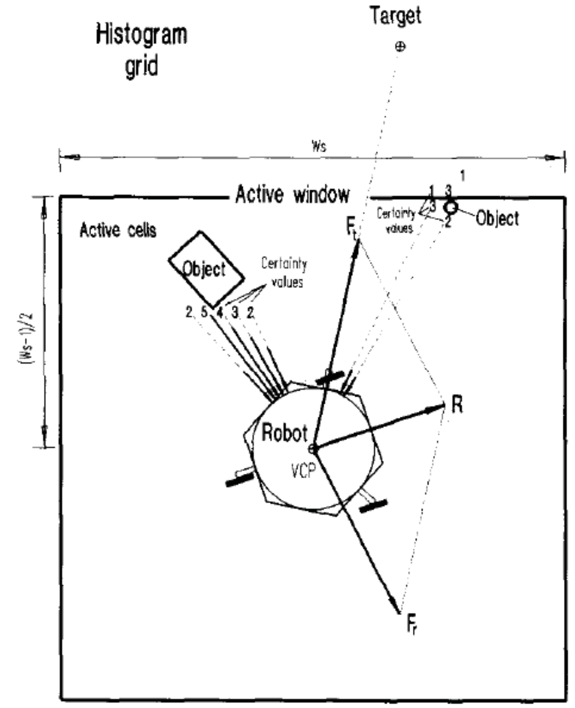

Histogram grid
Target
Ws
Active window
Active cells
Object
Certainty values
2 5 4 3 2
1 3 3 2 1
Certainty values
Object
F_t
R
Robot
VCP
F_r
(Ws-1)/2

Figure 2.4: Environment ©1991 IEEE

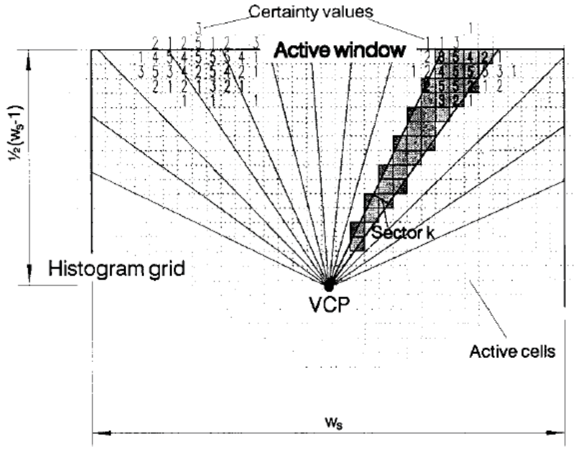

Certainty values
Active window
½(w_s-1)
Sector k
Histogram grid
VCP
Active cells
w_s

Figure 2.5: Cartesian Histogram Model ©1991 IEEE

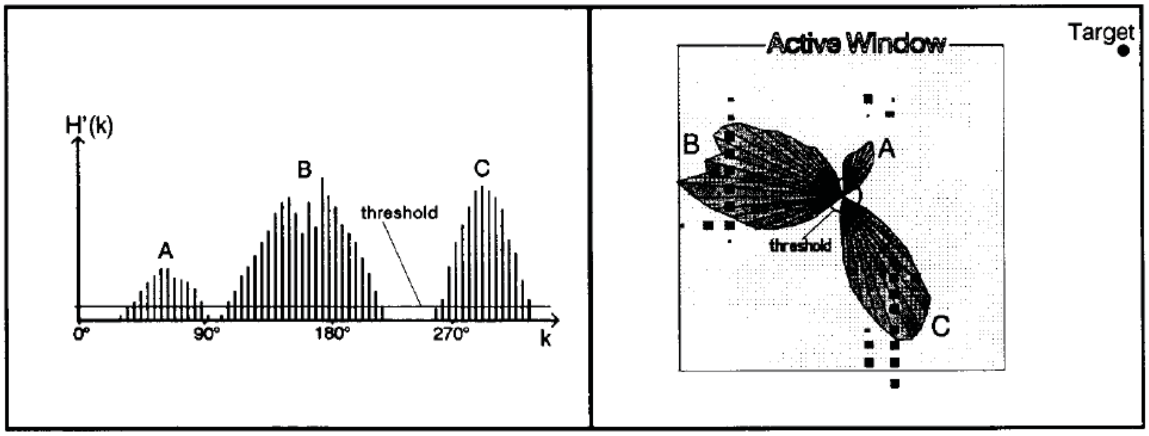


Figure 2.6: Polar density in Cartesian and polar form before applying threshold ©1991 IEEE

2.2.4 Dynamic Window Approach

This obstacle avoidance method chooses circular trajectories while considering kinematic and dynamic constraints of the robot for reliable obstacle avoidance [29, 30, 31]. In this approach, the robot searches through a set of tuples of translational and angular velocities. Fig. 2.7 shows an environment that will be translated into velocity space. Admissible velocities are extracted (Fig. 2.8) that allow the robot to avoid hitting the obstacle based on its current position, velocity, and applicable acceleration range.

Between filtered velocities, the one that can be achieved fastest considering current velocity and robot's acceleration is chosen (Fig. 2.9).

Unfortunately, this method can cause the robot to get trapped in local minimas since it is a local path planner. This algorithm is combined with a grid-based navigation algorithm

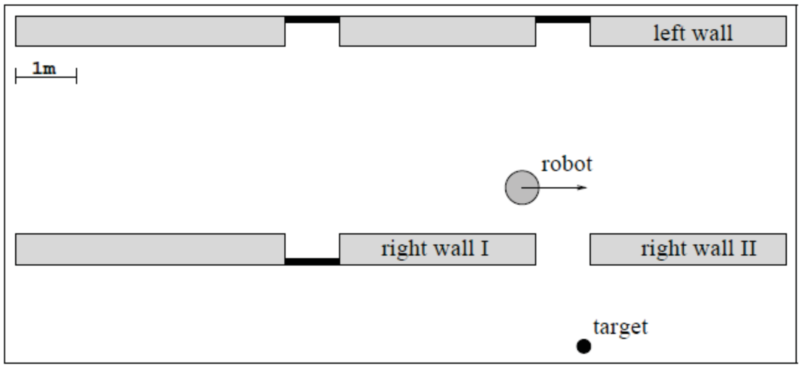


Figure 2.7: An example setting for dynamic window approach ©1997 IEEE

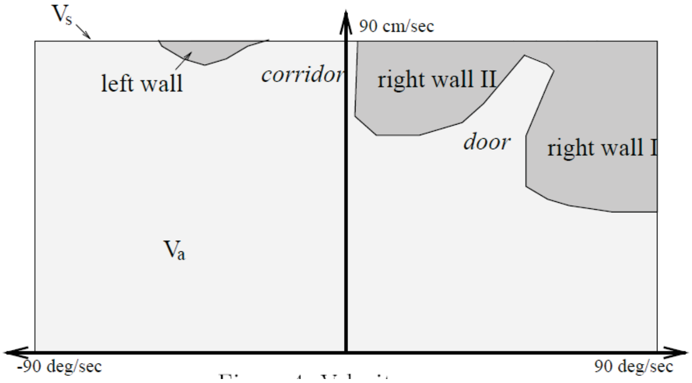


Figure 2.8: Admissible velocities ©1997 IEEE

resulting in an approach called "global dynamic window" [32], which can avoid traps but needs complete information about the environment in advance.

2.2.5 Elastic Band

Elastic band method [33, 34, 35, 36] involves simultaneous path planning and control for robot vehicles by combining global path planning and real time obstacle avoidance. The robot moves on a free path between start point and the destination that is deformed by

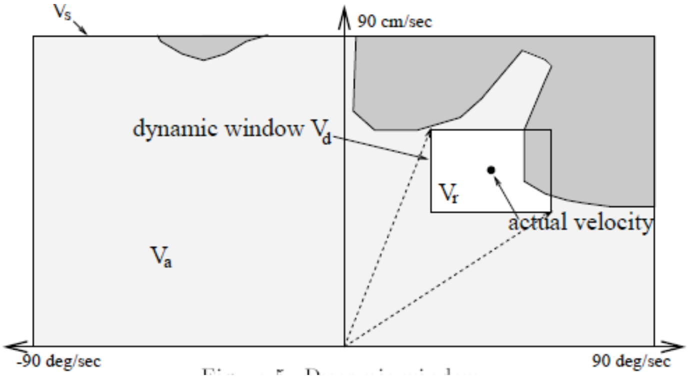
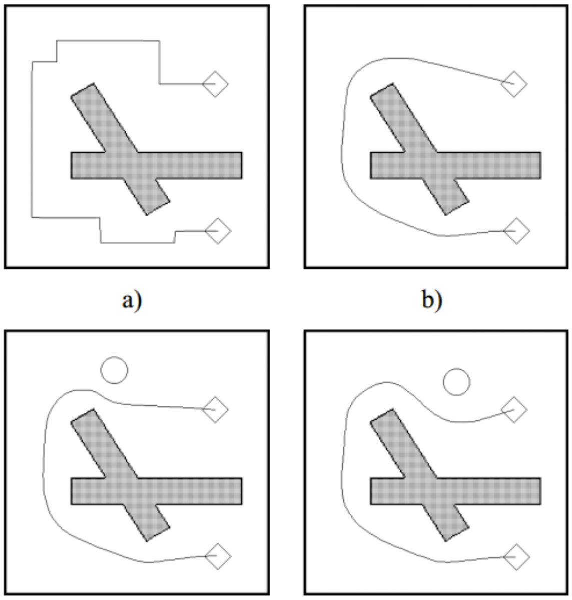


Figure 2.9: Dynamic window representation ©1997 IEEE

virtual forces from obstacles. The deformation is done in real time and results in a smooth path, which maintains clearance from all the obstacles in the environment. This path can change its shape and length based on the changes happening in the dynamic environment sensed by sensors mounted on the robot. This method uses a three-level hierarchy in which the first level is global path generation (Fig. 2.10). This path is then deformed based on the effect of two types of virtual forces. One type is the external repulsion forces from obstacles and the other type is internal contraction force of the elastic band. The external repulsion forces result in providing clearance from obstacles and counteract the internal force. The final stage is using control methods for path tracking and having the robot moving on the developed path.

This approach eliminates the need for doing the global path planning multiple times during the mission and only deforms the path according to changes in the environment. The downside of this method is unsuccessful performance when there are big changes in the environment.



a) b) c) d)

Figure 2.10: ElasticBand ©1993 IEEE

2.2.6 Potential Field

Potential field method is one of the popular local path planning methods due to its computational and mathematical simplicity [37, 38, 39]. Potential field has many modified versions and they are all based on the idea proposed by Khatib [37]. In this method, the robot is considered a point under influence of an attraction force from the destination and repulsion forces from obstacles. A conventional potential field is as follow:

$$U(x) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{\rho} - \frac{1}{\rho_o}\right) & if\rho \leq \rho_o \\ 0 & if\rho \geq \rho_o \end{cases}$$

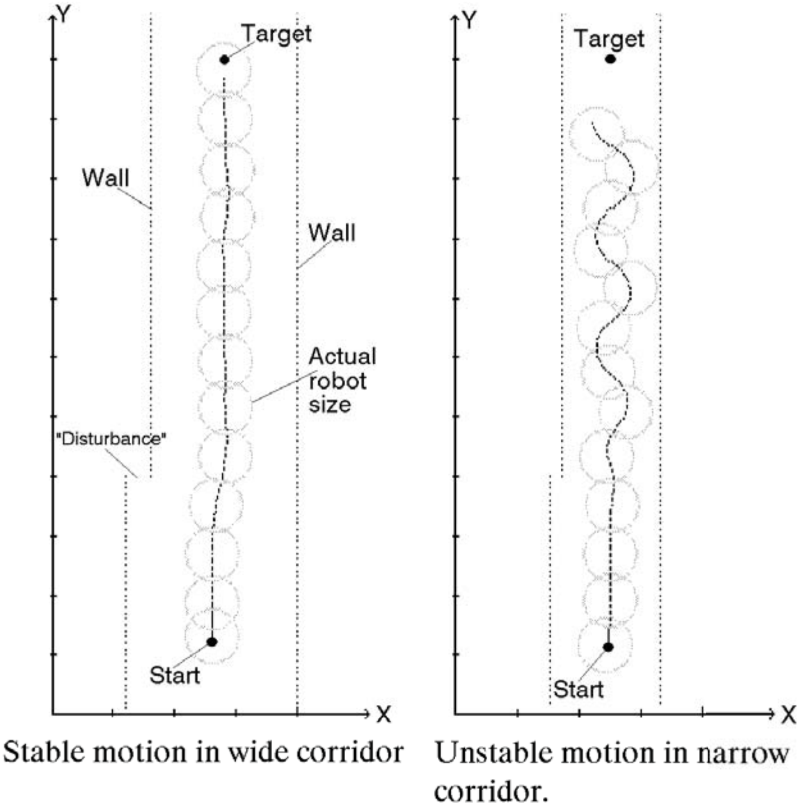

Y
Target
Wall
Wall
Actual robot size
"Disturbance"
Start
X
Stable motion in wide corridor
Y
Target
Start
X
Unstable motion in narrow corridor.

Figure 2.11: Unstable performance of potential field in narrow corridors ©1991 IEEE

In which ρ represents distance between the robot and obstacle and ρ_0 represents the influence distance of this potential field. η is a constant controlling the magnitude of repulsion force.

In potential field method, the destination is a global minimum in the environment and obstacles are peaks. The trade-off for the simplicity and efficiency of this algorithm is its undesirable performance in local minimas which lead to the robot getting trapped in narrow passages (Fig.2.11) that can result in unstable behavior.

2.2.7 Fractional Potential Field Method

Fractional potential field [42, 43] is a modified version of potential field method. In this approach the potential field is not only a function of relative position of the robot with respect to target and obstacles but also the relative velocity of these components define the potential value of robot. The attraction force between target and robot is calculated as follow:

$$U_{att}(p,v) = \alpha_p||p_{tar}(t) - p(t)||^m + \alpha_v||v_{tar}(t) - v(t)||^n$$

Where p and p_{tar} denote the positions of respectively the robot and the target. v and v_{tar} represent respectively the velocities of the robot and the target. The rate of change in potential is controlled by α_p, α_v, m, and n, which are positive parameters. The repulsion force is also a function of relative position and velocity:

$$U(x) = \begin{cases} 0 & \text{if } \rho_s(p,p_{obs}) - \rho_m(v_{RO}) \geq \rho_0 \text{ or } v_{RO} \leq 0 \\ \frac{\eta}{2}\left(\frac{1}{\rho_s(p,p_{obs}) - \rho_m(v_{RO})} - \frac{1}{\rho_o}\right) & \text{if } 0 < \rho_s(p,p_{obs}) - \rho_m(v_{RO}) < \rho_0 \text{ and } v_{RO} > 0 \\ \text{Not Defined} & \text{if } v_{RO} > 0 \text{ and } \rho_s(p,p_{obs}) < \rho_m(v_{RO}) \end{cases}$$

In which ρ_s is the shortest distance between the robot and the obstacle, ρ_0 is a positive value that controls the distance from the obstacle that starts influencing the potential. η is also a positive constant and can be used for adjusting the influence range of the obstacle's potential field. v_{RO} is the relative velocity of robot with respect to an obstacle, meaning if this value is positive, the robot and obstacle are getting closer to each other and vice versa. $\rho_m(v_{RO})$ is the distance to be traveled considering maximum applicable acceleration of robot that allows the relative velocity to drop to zero:

$$\rho_m(v_{RO}) = \frac{v_{RO}^2(t)}{2a_{max}}$$

This work was again extended in [44] to add the acceleration parameter to the potential value. One should keep in mind that estimation of velocity, and especially acceleration of moving obstacles (tangential and angular) in an unknown environment, needs very robust algorithms and accurate sensors. Another study on this type of path planning can be found in [45].

2.3 Robust Path Planner

This algorithm is a geometric path planner to give the most optimal and stable performance with respect to limited information about the environment. To be able to analyze and improve the control algorithm, a simulation environment is developed to accommodate flexibility for testing. The simulation environment consists of different parts, starting with a model of the surrounding of the robot. It also includes models of sensors used by the robot to understand the surrounding (such as range sensor). Information received from the sensors is then fed to the control system. The control system by itself consists of two parts, one is decision making on behavior of the robot (path planning) and the other one is calculating the actions needed to achieve desired states by the robot considering its characteristics (path tracking). The outputs of control system are forces produced by motors and then motion of the vehicle. The simulation of the vehicle's position and orientation through the simulation time is the final result used for analysis and comparisons.

2.3.1 Modeling

In this section, different models used in the simulation will be discussed. These models are the means of interaction between the control systems of ground vehicle and environment and are needed to test and analyze the performance of the path planning algorithm. The main goal during development of these models was that they should be general enough to support different types of components (e.g. different sensors, vehicles, obstacle shapes and sizes, etc). Also, the graphical visualization method for each component will be discussed separately.

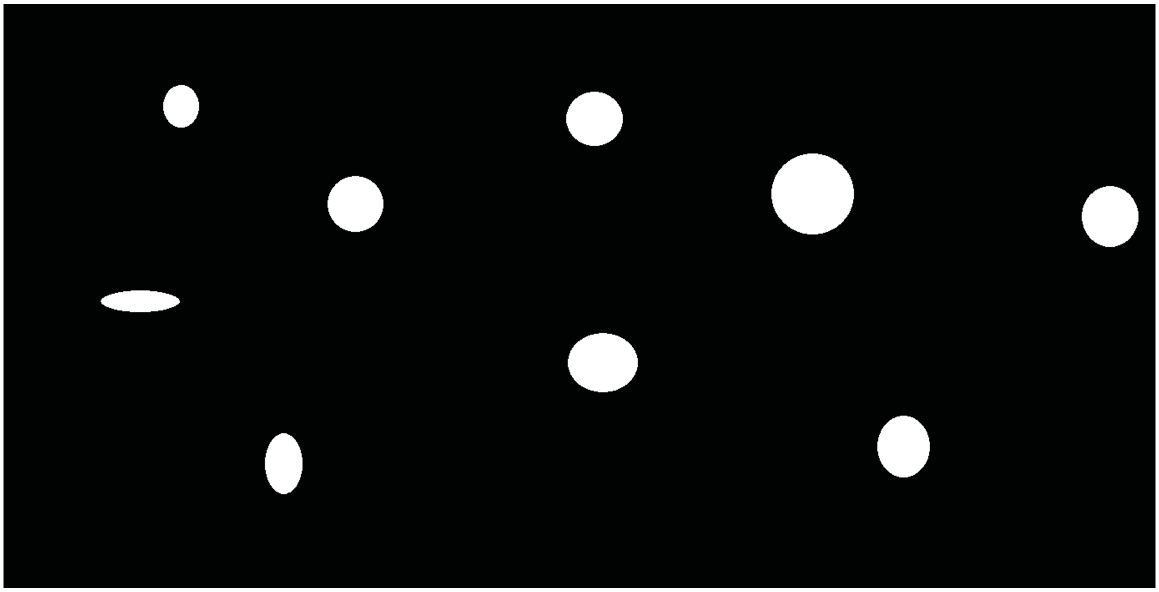


Figure 2.12: Environment representation

Environment: The simulation uses black and white images provided by the user represent the test setup and present outputs on which all the other information is shown. The white pixels in the image represent obstacles and the black pixels are empty spaces. These images can be made in any graphical editor software and be imported in to the simulation as long as they are black and white. At any point during the simulation, overlap of the vehicle's body and white pixels (obstacles) indicate collision.

Proximity sensor: This part of the algorithm models proximity sensors. This type of sensors are used to measure the distance from obstacles they are facing. This algorithm is capable of measuring one or two dimensions for generality purposes. Two dimensional sensors can output distances sensed in multiple angles at the same time. For modeling the performance of a two dimensional sensor, lines are drawn starting from the sensor and extended outward until reach an obstacle (white pixel). These lines are limited by the

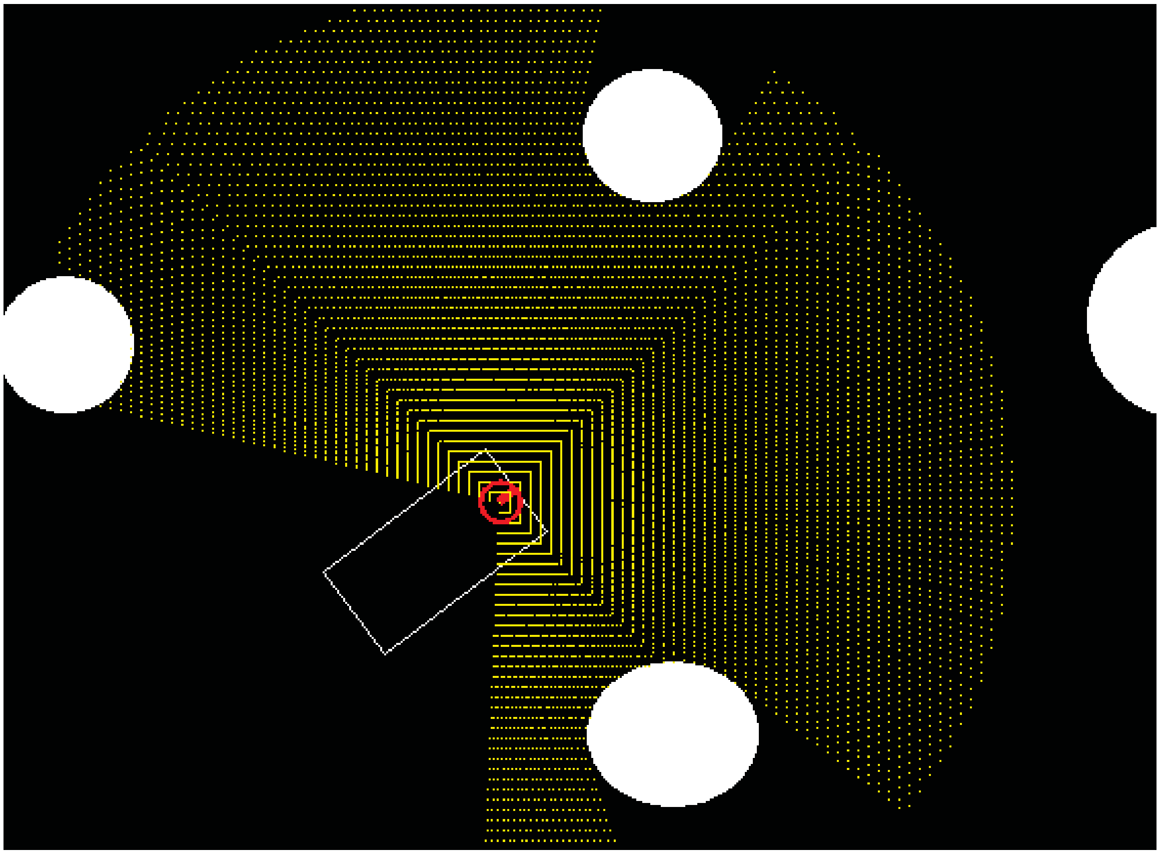


Figure 2.13: Two dimensional sensor representation

configurable distance range and angular range, also they and are distributed according to angular step from model's configurations. By decreasing the angular range of the sensor to zero, the algorithm can be used to model an one dimensional proximity sensor. An array consisting of the length of each line and an array of corresponding angles are the products of proximity sensor modeling algorithm.

Motor Controller and Vehicle Dynamics: Input of motor controllers is usually in the form of velocities of left and right wheels. The motor controller calculates the difference between commanded tangent velocity and current tangent velocity of the vehicle and also

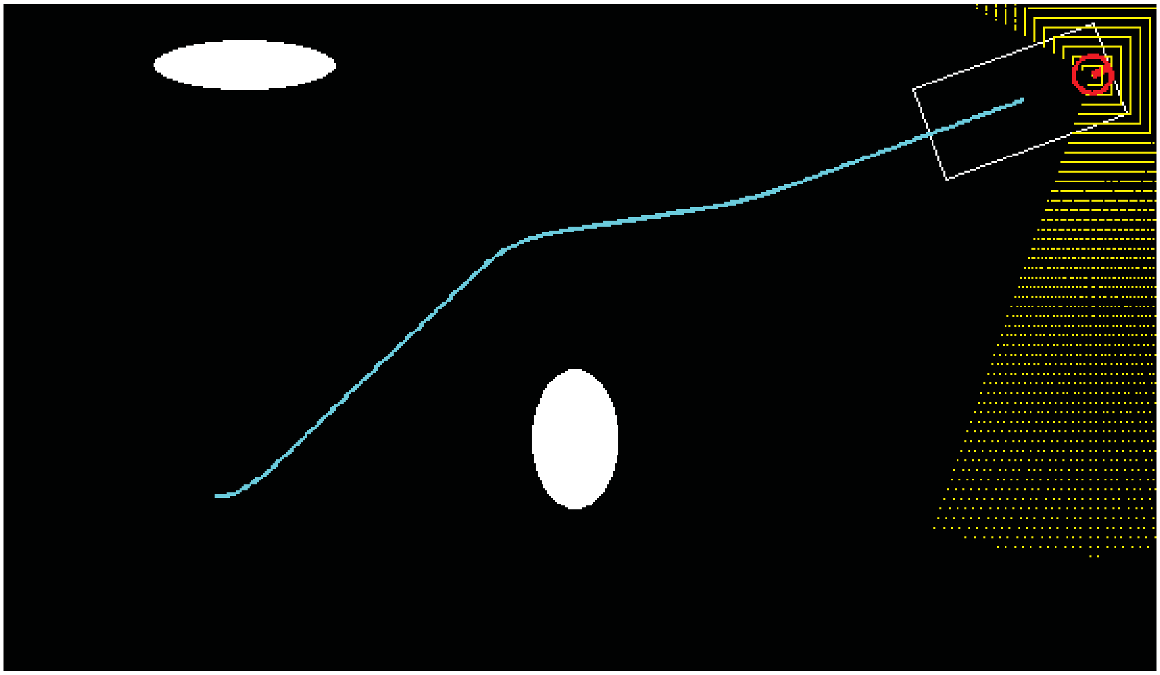


Figure 2.14: Path representation

considers the difference between commanded angular velocity and current angular velocity. The output will be two forces applied to the body of the vehicle, one on the right side and the other on the left side. These forces are proportional to the mentioned calculated errors. Using Newton's law, tangential and angular accelerations of the vehicle are calculated. The future position of robot for each time step can be calculated from these accelerations along with current tangential and angular velocity, position, and orientation of the robot.

All the modeling components mentioned in this section equip the robot to make decisions based on the environment and the visualization method of each component helps us evaluate and compare the performance of the control algorithm.

2.3.2 Algorithm

The reason for developing a simulation environment was to provide us with enough tools for developing a path planning algorithm. This algorithm is mainly expected to be capable of performing in an unknown environment. The main challenge in such environment is inability to achieve a pre-calculated path. The robot has to avoid each obstacle when it is in the robot's sensing range in real-time. Moreover, the limitations of implementing the developed algorithm on a real vehicle should be considered. To reach this, the path planning algorithm needs to only rely on the mentioned models of sensors and actuators and be a sensor-based algorithm. Also, it should be simple enough to support real-time calculation on the robot's embedded system, since planning a newly introduced obstacle should be done with minimal risk. This simplicity should not counteract the autonomy of the system and its independence of human intervention. In the end, it needs to be capable of performing in cluttered environments with different obstacle sizes and shapes.

This path planning method is based on analyzing the environment to determine the optimal direction to move at any given time according to the position of the vehicle, obstacles, and target. To achieve a solution close to optimal, it is expected to use all of the potential of range sensor and consider all the obstacles as soon as they enter the range of the sensor unlike other algorithms that only change their path based on obstacles in close proximity. The algorithm also shows stable performance in narrow corridors and can handle local minimas. The process of using sensor outputs and generating paths can be divided into multiple steps:

Obstacle Segmentation: The algorithm receives data from proximity sensor and translates it into a set of obstacles for further analysis. All the elements of the distance array received from proximity sensor that do not reflect any reading and their corresponding angles are filtered out. Then the filtered arrays are translated from polar to Cartesian reference. All the readings that have a distance less than a safe distance with their neighboring readings are grouped into one obstacle. Two obstacle segmentation examples are illustrated in Fig. 2.15. The safe distance is calculated based on the dimension of the robot which is 1.2 times of the robot's diagonal length. The output of this step is a list of left side and right side direction of all the segmented obstacles.

Calculation of subgoal candidates: The next step is calculation of two safe directions for each obstacle, one on the right side and one on the left side. Each subgoal is positioned along one of these directions. Figure 2.16 shows the method of calculating these directions. From the robot's perception, all the larger obstacles are constructed from these small obstacles. So for a large obstacle, the same calculation is done for each reading in the group and the out-most right and the out-most left are considered as final subgoal candidates and all the subgoals positioned in between are discarded. The result will be a safe direction to move (any of the two final subgoals).

Selection of best subgoal: A method of comparison is employed to find the best direction

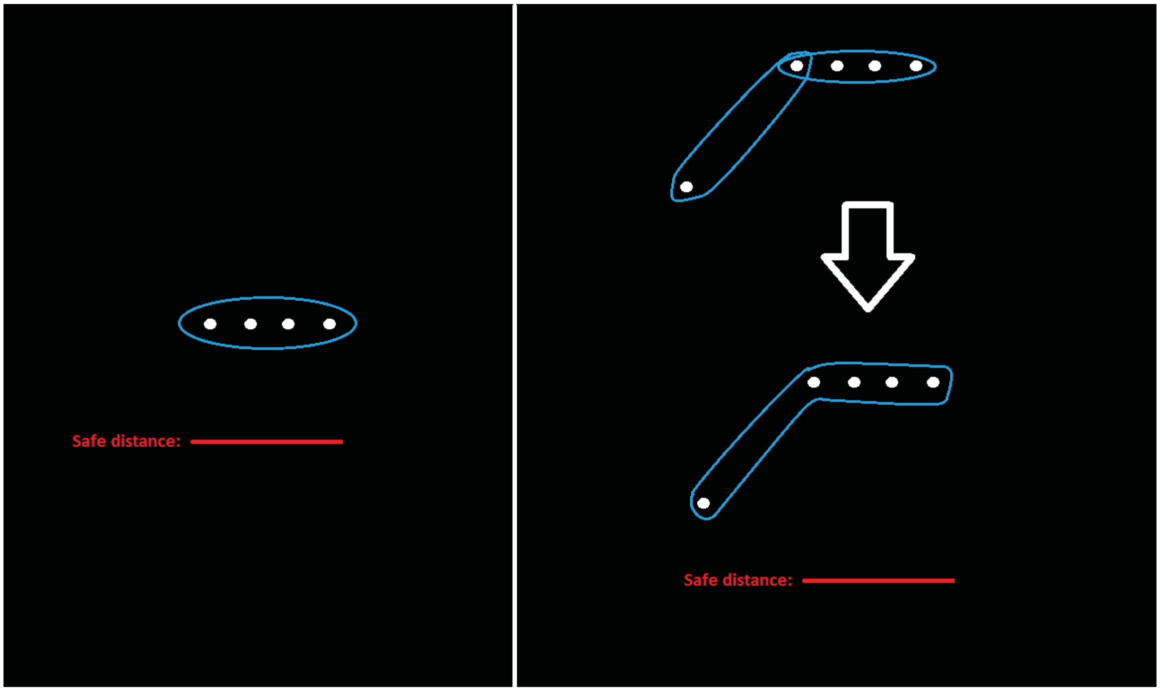

Safe distance:
Safe distance:

Figure 2.15: Obstacle Segmentation

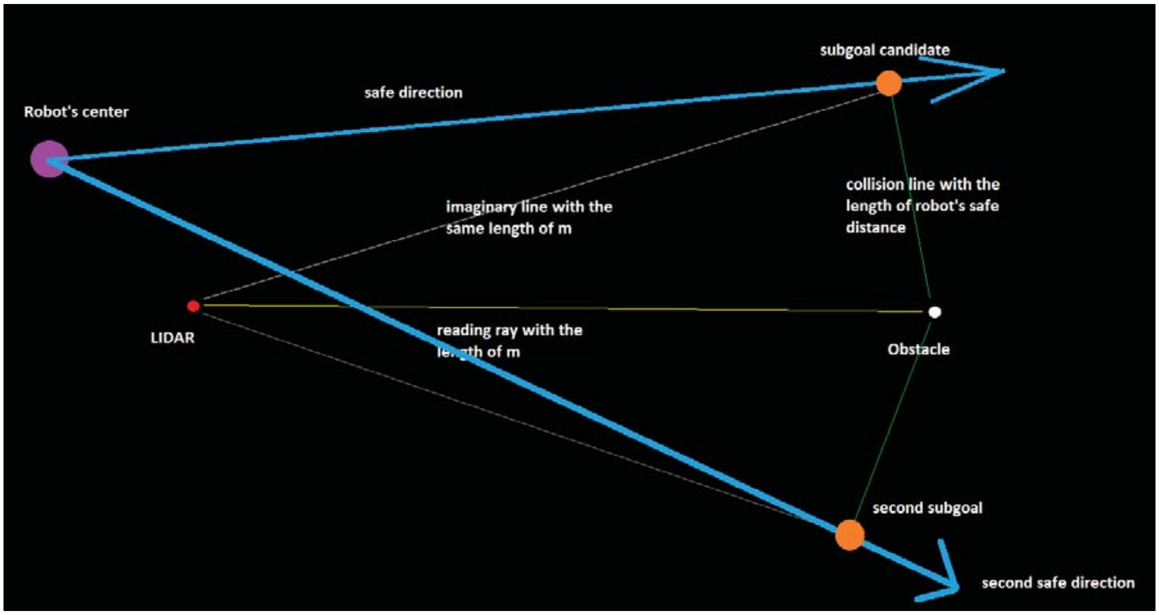

subgoal candidate
safe direction
Robot's center
imaginary line with the same length of m
collision line with the length of robot's safe distance
LIDAR
reading ray with the length of m
Obstacle
second subgoal
second safe direction

Figure 2.16: subgoal candidates

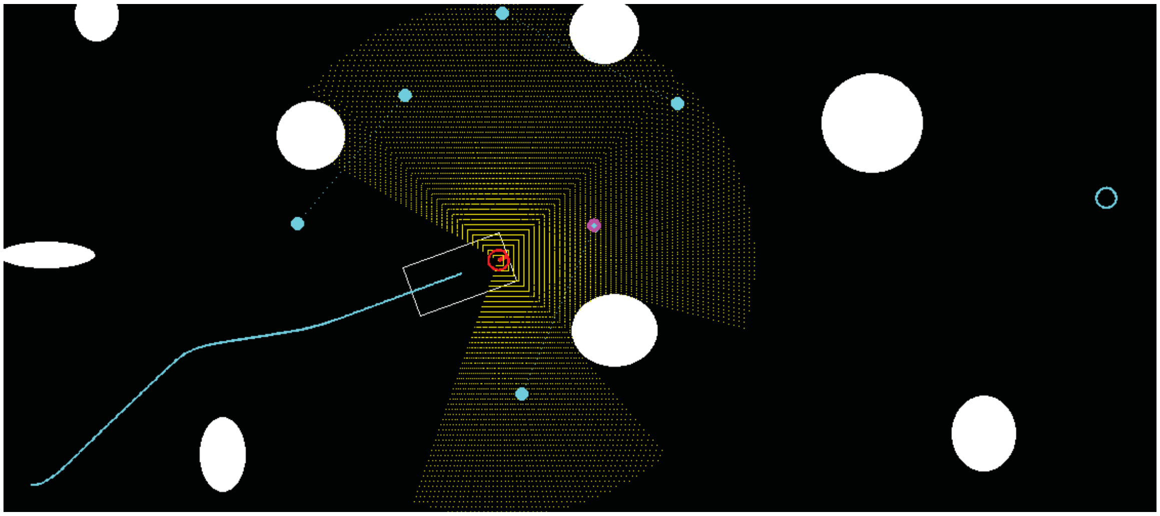


Figure 2.17: Subgoals candidates and the best subgoal: Best candidate has a purple highlight, final destination is the unfilled blue circle on the right side

from all the subgoals of all the obstacles. In this part of the algorithm, the ultimate destination of the robot is set as the initial subgoal. The robot will start analyzing this goal, if the straight path towards this goal is not between the left and right subgoals of any of the obstacles then the robot starts moving towards it. But if the subgoal is between safe directions of an obstacle, one of the subgoals around that obstacle with least difference from the original goal is selected as the new subgoal. For some instances, the new subgoal is also blinded by another obstacle. For this reason, the algorithm is repeated again until no more changes in selected subgoal is made.

In conclusion, the algorithm uses all of the sensor's potentials, meaning as soon as an obstacle is in sensor's field of view, it will potentially affect the calculation of robot's path.

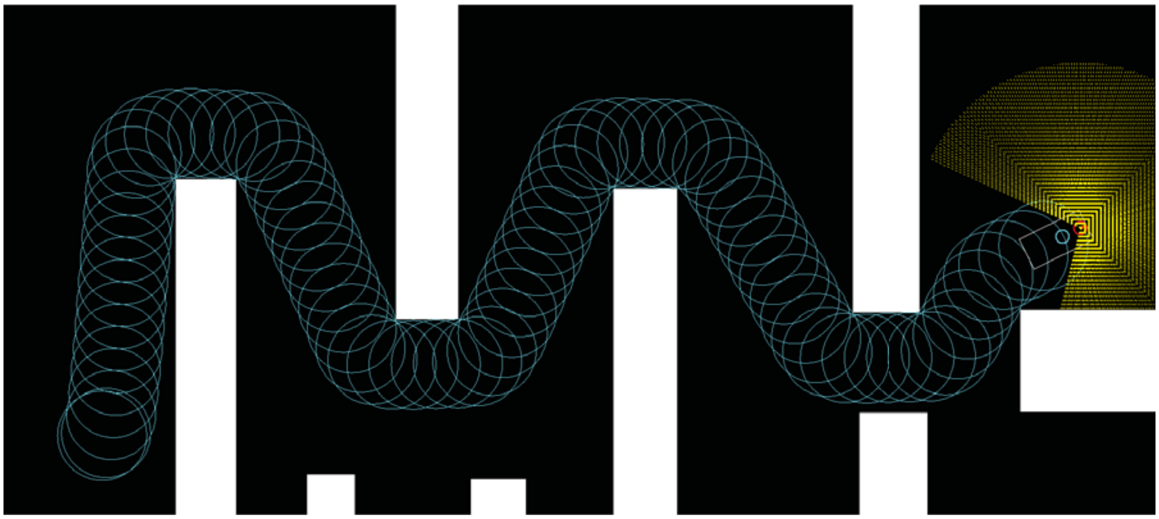


Figure 2.18: Test results of robust path planning algorithm in an environment similar to indoors

2.4 Robust Path Planning Algorithm Results

Figures 2.18 and 2.19 show the performance of the robust algorithm detailed in 2.3. It can be seen that the robot avoids obstacles with minimal divergence from the shortest path. In Fig. 2.18 robot finds the only path towards the goal in first try, and also is able to show satisfactory results moving in very narrow spaces. Fig. 2.19 is the same algorithm in an environment with multiple possible paths where the robot chose the shortest path while keeping enough clearance from obstacles.

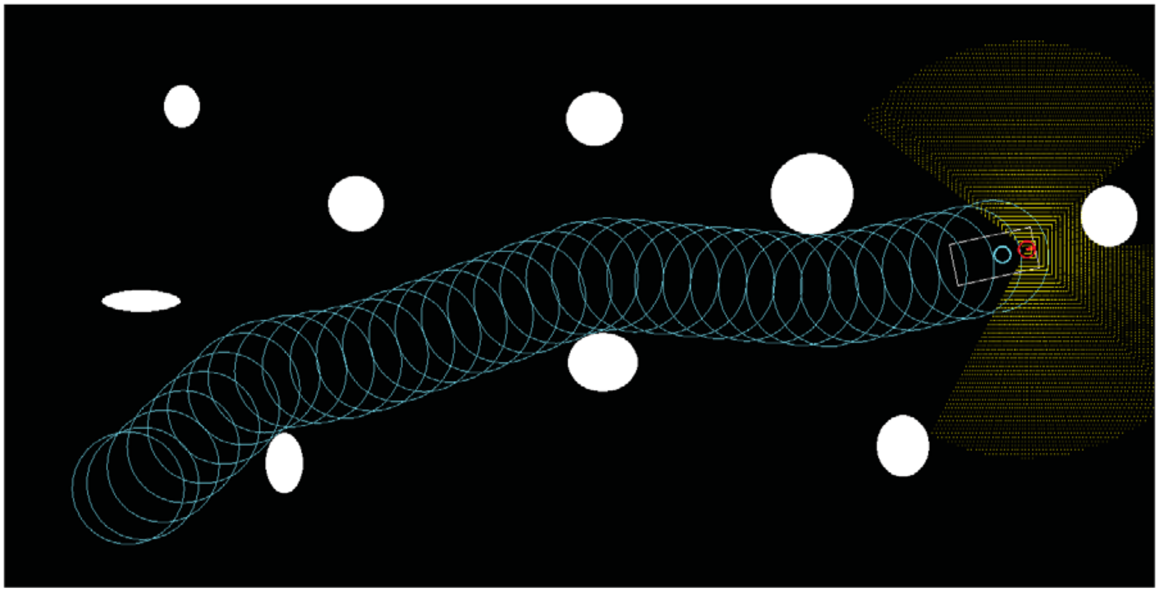


Figure 2.19: Test results of robust path planning algorithm in an environment similar to outdoors

Chapter 3

Navigation for Power Distribution Systems

For completing an autonomous mission, three processes are associated with navigation of the machine: mission planner, path planner, and path tracker. The first process in mission planner, translates the objectives of the mission to assumptions of the path planner. For example the mission planner sets the path planner's destination position based on the requirements of the mission for moving to different targets one after another. During the next process, the path planner method uses the positions (also, in some methods, directions) of start point and destination and generates a path via an algorithm. Different types of path planning methods were detailed in chapter 2. The final process of navigation is the path tracking, which outputs actuator commands by considering the system's dynamic characteristics and use of a controls system to follow the path planner's chosen trajectory with minimal error.

In this chapter, a navigation solution developed for autonomous power distribution proof-of-concept is detailed.

3.1 Multi-layered Control Hierarchy Structure

A multi-layered hierarchy has shown to be a practical method of administrating an autonomous mission [46, 33] That can be configured to deliver redundancy and ease the way for modifications.

In this project, a tailored mission control hierarchy with multi-feedback (Fig. 3.1) has been implemented on robots to provide on-site autonomous assessment, generate safe paths, and facilitate physical connection. The process is developed to be distributed and decentralized, with minimal communication requirement, and general enough for use in a real-size system with different perception modules and different sensor suites.

Each block in Fig. 3.1 carries out a specific task. Mission planner divides the mission into multiple tasks and decides which task should be completed at each moment. While performing the mission, the mission planner decides which target to switch to. This is

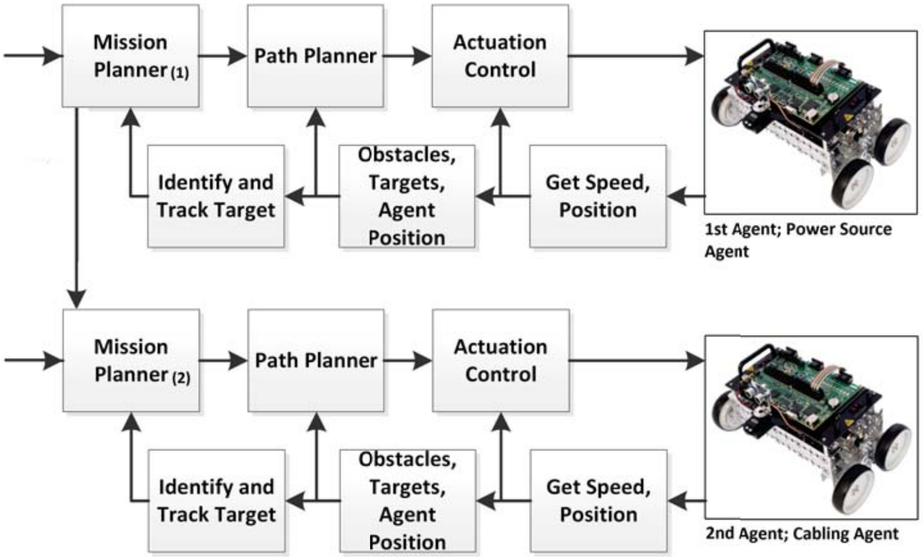

Mission Planner (1)
Path Planner
Actuation Control
Identify and Track Target
Obstacles, Targets, Agent Position
Get Speed, Position
1st Agent; Power Source Agent
Mission Planner (2)
Path Planner
Actuation Control
Identify and Track Target
Obstacles, Targets, Agent Position
Get Speed, Position
2nd Agent; Cabling Agent

Figure 3.1: Control hierarchy of 2 agents

done based on the information gathered from the robot and environment to accomplish the mission goal. The mission planner sends the decided target information to the path planner one at a time. The path planner algorithm decides on the future position and direction of the robot based on the position of the introduced target, current position of robot, and visible obstacles. Actuation control achieves these states using a feedback controller and sends signals to hardware. Speed and position of the robot is fed back to actuation control to compensate for the error in the robot's behavior. A detailed description of each block can be found in Sections 3.2-3.4.

Along with having the mentioned hierarchy implemented on both robots, a decision regarding the method of communication between two robots was needed. In a control hierarchy, deeper algorithms like actuation control need to be updated and the results of these algorithms need to be monitored at higher rates than top-level algorithms like mission planner.

On the other hand, having continuous and simultaneous communication cannot be guaranteed in different mission environments. Hence it is preferred to have most of the control hierarchy levels independent from information sent from other agents. To accomplish this, it was decided to only have communication on the mission planner level. The down side of this decision is having no centralized path planner, which can lead to requiring more complicated algorithms to overcome dynamic obstacles. But it is possible to get to a solution by aid of the dependent mission planner. Details of the extended mission planner for each robot will be discussed in the following subsection.

3.2 Mission Planner

This high level algorithm is developed to control the stages of accomplishing the power distribution mission. As described in section 3.1, the mission planner assigns the robot with sub-goals to achieve the overall mission goal. In other words, from the mission planner point of view, the robot is climbing a ladder of sub-goals step by step. Having two agents in a mission with different tasks requires that the mission planner of each agent to allocate a set of sub-goals that are defined based on the role of the robot and its objective.

For the purpose of this work, mission planners of robots are designed to execute the following steps:

1. The power source agent moves towards the imaginary point in front of one of the power loads. This power load can be chosen based on priority or based on initial distance of the power source agent or the mixture of two.
2. The power source agent moves towards the contact point of the first power load. At this point the first load is powered up.
3. The power source agent informs the cabling agent to start its mission.
4. The cabling agent moves towards the imaginary point in front of the second electrical contact point on the power source agent.
5. The cabling agent reaches the second electrical contact point and make the connection.
6. The cabling agent then moves towards the imaginary point in front of the second power load.
7. The cabling agent reaches the electrical contact point of second power load and makes the connection.

The reason for having an imaginary target in front of the connection points is to ensure proper facing while making the electrical connection.

Based on the proposed scenario, the second robot should start moving towards the connection agent as soon as the connection agent finishes its final goal (i.e. making a physical connection with power load). For communication between agents on this level, the mission planner of the first robot transmits a Boolean value for the successful accomplishment of its final goal. The second robot monitoring this value continuously and as soon as it realizes that first agent has accomplished its goal, it starts moving towards the first agent and makes a physical connection with it in the same manner that the first agent connected to the first power load.

3.3 Simple Path Planner

This algorithm was developed for experimental validation of autonomous power distribution system on lab platforms in unknown environments.

The idea of having an unknown workspace requires repeated execution of the path planning algorithm to facilitate robust avoidance from moving obstacles or late recognized obstacles. Moreover, system should tolerate errors caused by environmental noises and disturbances.

The flowchart of path planning algorithm is illustrated in Fig. 3.2. The process starts by receiving a target location from the mission planner, then it will make a straight path to reach the target point and command the agent to face the target. The agent starts following the line and at the same time gets feedback from the environment through IR sensor readings. Sensing obstacles through the feedback alters the original direction of movement. In the

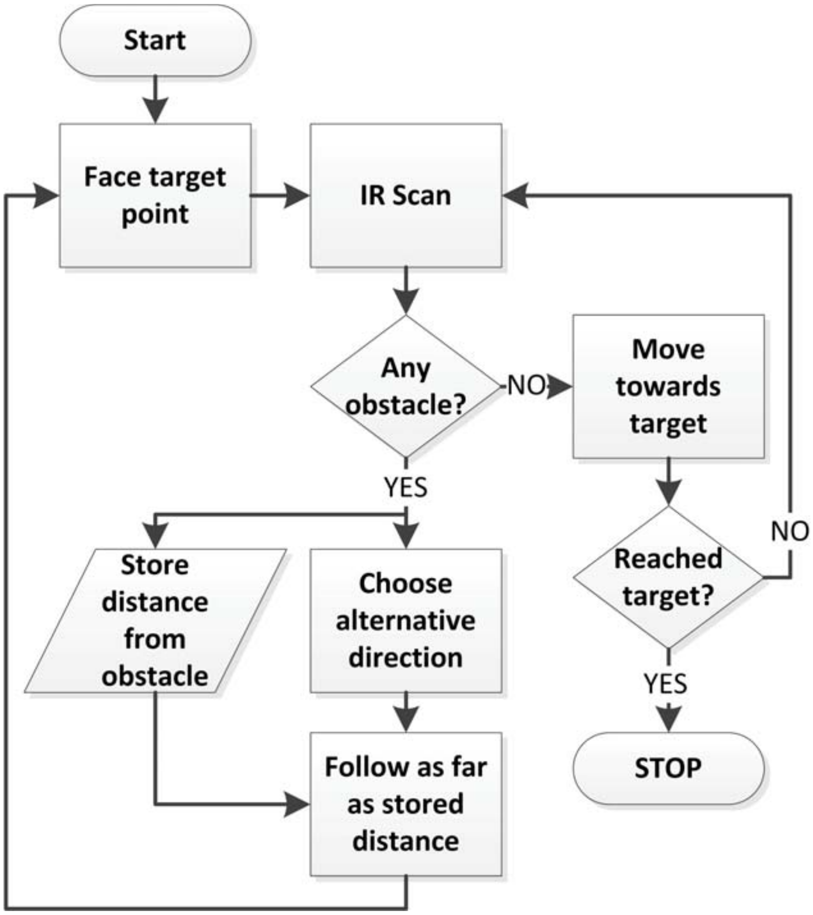

Start
Face target point
IR Scan
Any obstacle?
NO
Move towards target
YES
Store distance from obstacle
Choose alternative direction
Reached target?
NO
YES
Follow as far as stored distance
STOP

Figure 3.2: Path planning and target reaching algorithm flowchart

feedback process, the safest direction in front of the robot is chosen as the feedback value. If this direction is different from the direction towards the target (i.e. there is obstacles in front of the robot), the path planner algorithm stores the distance value from the closest IR reading and commands on moving in the suggested (alternative) direction. The new direction will be followed as far as the stored distance value. After reaching the end of the alternative line, the agent faces the actual target point again and repeats the process until it reaches the destination.

As Fig. 3.3 shows, each IR reading goes through a Butterworth filter in order to reduce

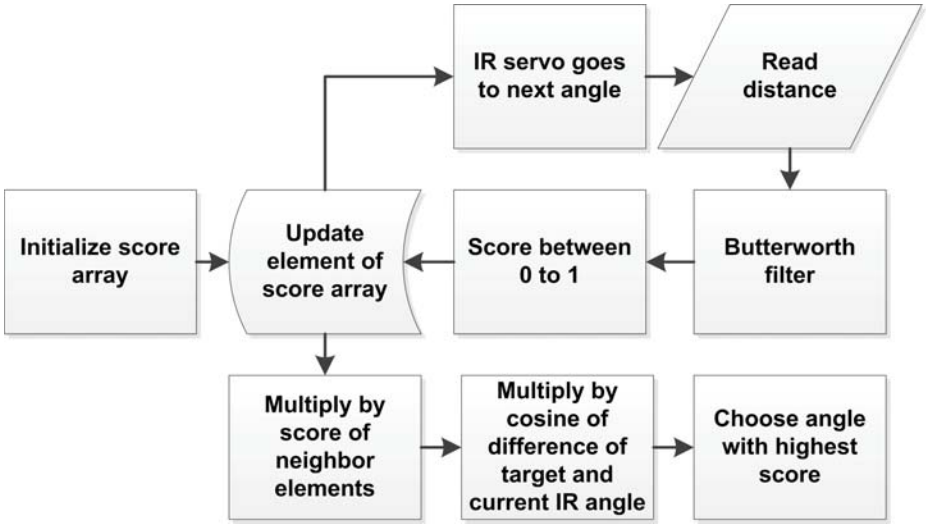

IR servo goes to next angle
Read distance
Initialize score array
Update element of score array
Score between 0 to 1
Butterworth filter
Multiply by score of neighbor elements
Multiply by cosine of difference of target and current IR angle
Choose angle with highest score

Figure 3.3: Direction choosing algorithm using IR scan

noise. The filtered data then goes into the direction-choosing algorithm. Based on this algorithm, each angle in the scanning area gets a score between zero and one. If the IR reading for an angle is less than a constant (e.g. 100 cm), it gets a score of zero and if it is more than a larger constant (e.g. 150 cm), it receives a score of 1. All the readings in between get scores based on linear interpolation. All these scores are put into an array that has an element to represent each angle. In the next step each score (element) is multiplied with its neighbors (e.g. 25 neighbors on each side), which allows the algorithm to choose a direction with neighbors of equally good scores. Then the score of each angle is multiplied with the cosine of the angular difference with the target's direction so that the algorithm has a tendency to choose a direction close to the target's direction, but at the same time this direction has a reasonable distance from the obstacle due to multiplication by its neighbors. The number of affecting neighbors is dependent on the accuracy and scanning rate of the proximity sensor and geometrical properties of the robot. In this case, accuracy and scanning rate had the most effect on the risk of crashing into obstacles, so the number of affecting neighbors was chosen based on trial and error only. A more reliable proximity sensor equipped with an algorithm that considers the geometric properties of robot can result in choosing directions that not only avoids obstacles but also eliminates paths that are not time efficient and are too far from obstacles.

3.4 Path tracking

A feedback controller was designed using the first order kinematics model of the vehicle. The mathematical model of robot's kinematics can be described as follow:

$$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta}_c \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ w \end{bmatrix}$$

where v and w are tangential and angular velocities of robot and x_c, y_c and θ_c are position and direction of vehicle in the workspace. A trajectory can be defined as a function of time like $(x_r(t), y_r(t))$ using the inverse kinematic equation and this trajectory can be translated to the robot inputs for feed-forward control:

$$v_r(t) = \pm\sqrt{\dot{x}_r^2(t) + \dot{y}_r^2(t))}$$

$$w_r(t) = \frac{\dot{x}_r(t)\ddot{y}_r(t) - \dot{y}_r(t)\ddot{x}_r(t)}{x_r(t), y_r(t)}$$

This kind of mobile vehicle usually takes wheel velocity as input, which can be derived from feed-forward control inputs mentioned earlier, $v_R = v + \frac{wL}{2}$ and $v_L = v - \frac{wL}{2}$, where v_L is the velocity of left wheel and v_R is the velocity of right wheel.

Defining errors as differences between reference position of the robot on the path and the robot's actual position is the first step in forming the feedback controller.

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix}$$

After linearizing around an operating point ($e_1 = e_2 = e_3 = 0, v_1 = v_2 = 0$), feedback values can be calculated from:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} -k_1 & 0 & 0 \\ 0 & -sign(u_{r1})k_2 & -k_3 \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix}$$

A proportional feedback controller is used and the gains are tuned based on trail and error considering hardware limitations, algorithms, and data transfer rates, which can have great effects on stability of the robot.

This method of path tracking and actuation control has shown satisfactory results in previous experiments for different applications. The same method is used for path tracking component in control algorithm of the simulation. The method does not eliminate the errors of nonlinearities of the vehicle but it is simple enough to keep running in a loop in high frequency and compensate for these errors.

Chapter 4

Power Distribution Experimental Results

4.1 Scenario

An example system that can benefit from autonomous power distribution systems is a communication tower in remote places. These devices demand three-phase 208 V_{ac}, so the robots need to fulfill this demand through power sources, line connections, and electrical power conversion [47]. While performing power distribution for such systems, there might be additional power needs by systems that use dc voltage such as telephony systems [48]and also there exist other devices that need a combination of ac and dc voltages [49].

Different power requirements during one mission call for an on-board conversion system, capable of converting power from the source to other forms needed by all these devices [50].

We consider a multitude of disaster scenario that require power restoration as a proof of concept. This concept requires extensive coordination of many distinct research disciplines, including path planning and coordination of autonomous vehicles [51, 52, 53, 54, 55] , power electronics and microgrids [56, 57], and disaster impacts and response [58, 59, 60].

In such scenarios, a small pallet or cargo containing an autonomous robotic microgrid system could be dropped by helicopter. It would then restore power quickly, requiring minimal human intervention. The autonomous robots will first establish a physical connection with the on-sight resources and loads. Then the on-board power electronics [50] will do the proper conversion of one form of electricity to another. The general steps that the system will take are:

1. Robot assets arrive and assess the power requirements of the local system.
2. Robots autonomously physically connect sources and loads into a microgrid structure.
3. The on-board power electronics convert source energy to a common distribution level.
4. Load-connected robots convert the distribution voltage to the needs of the load.

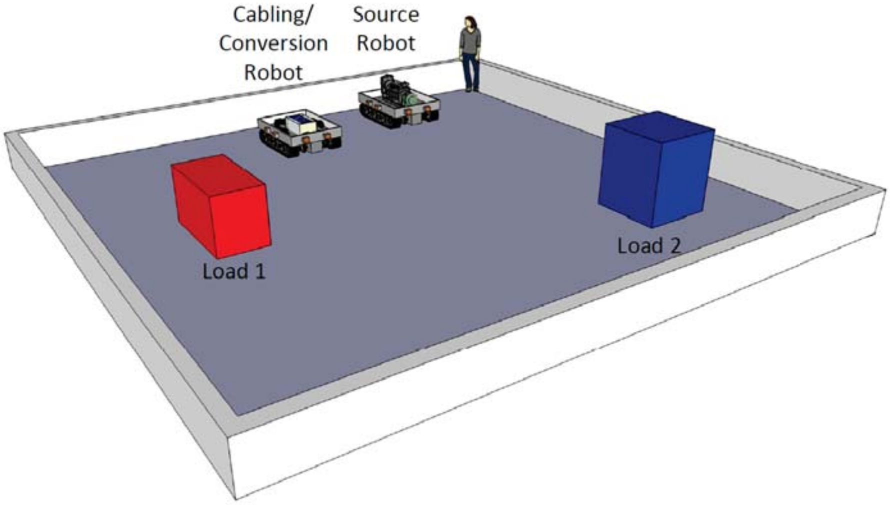


Figure 4.1: Initial state of power source robot, cabling robot, and two power loads

5. Robots re-configure system as energy assets and load change.

For proof-of-concept a scenario comprised of two autonomous microgird robots was chosen (Fig. 4.1); a power source robot, a cabling robot, and two stationary power loads. The power source robot can be representative of a mobile photovoltaic array, diesel generator system, or simply a battery, and the power loads could be representative of a hybrid vehicle charging station, or a communication tower. Fig. 4.1-4.4 show a step-by-step isometric interpretation of a power distribution mission implementation. Once the connections are established, the power source will start to deliver power to the loads.

For implementing and study of the performance of navigation and performing the proof-of-concept, some supporting components are needed. Section 4.2 presents these remaining components required for performing a lab-scale autonomous power distribution mission.

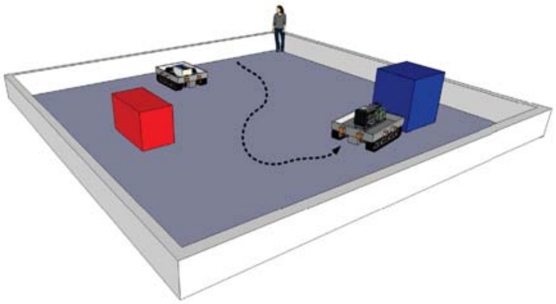


Figure 4.2: power source robot moves towards one of the power loads (blue) and makes an electrical connection

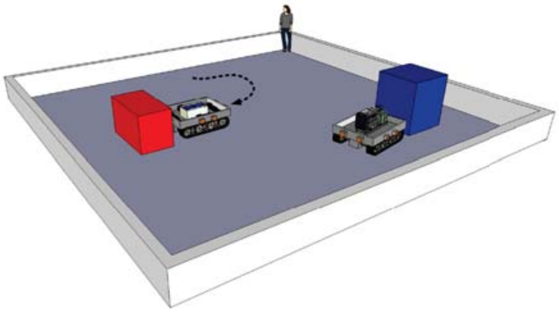


Figure 4.3: Cabling robot moves towards the other power load (red) and makes an electrical connection

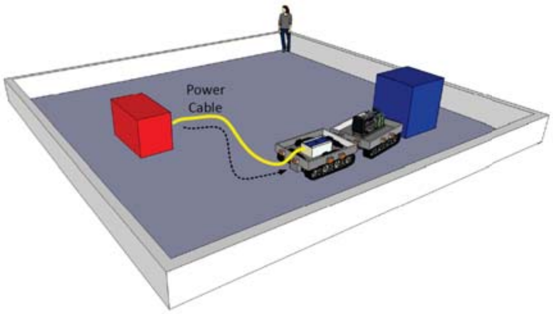


Figure 4.4: Cabling robot moves towards the power source robot and makes the final connection

Section 4.3 gives an explanation of the results of simple path planning algorithm accompanied with supporting hardware.

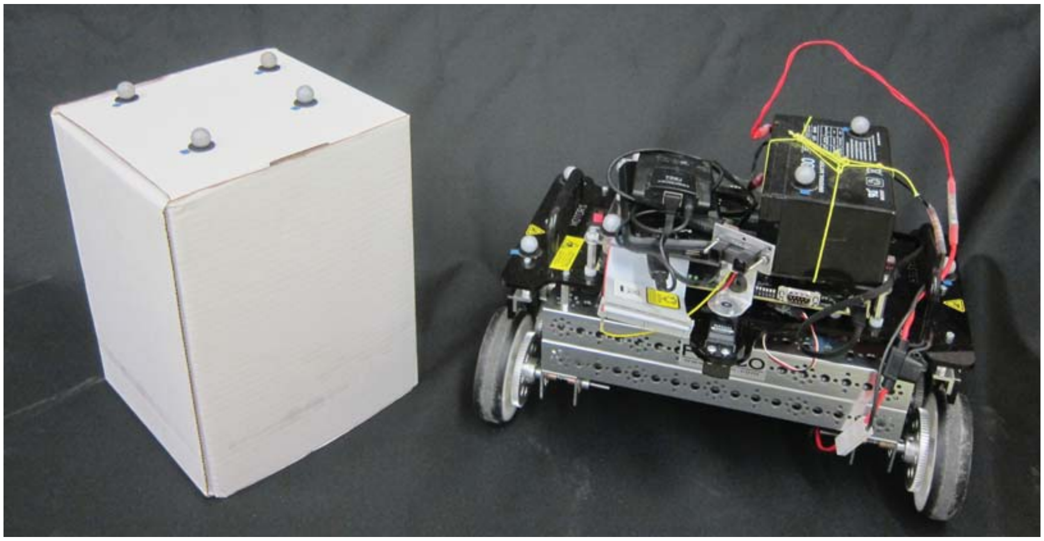


Figure 4.5: A Microgrid Agent and a Power Load; Photo by author

4.2 Hardware

In this section, required hardware for performing an example scenario for autonomous power distribution is presented. Fig. 4.5 shows some of the main components for the power distribution mission.

4.2.1 Power Electronics

The main element in development of autonomous power distribution systems is equipping robots with power modules that provide them with the capability to act as microgrid nodes. A microgrid is a collection of energy resources on a common network. These resources include generation, conversion, loads, and energy storage devices [15]. The microgrid

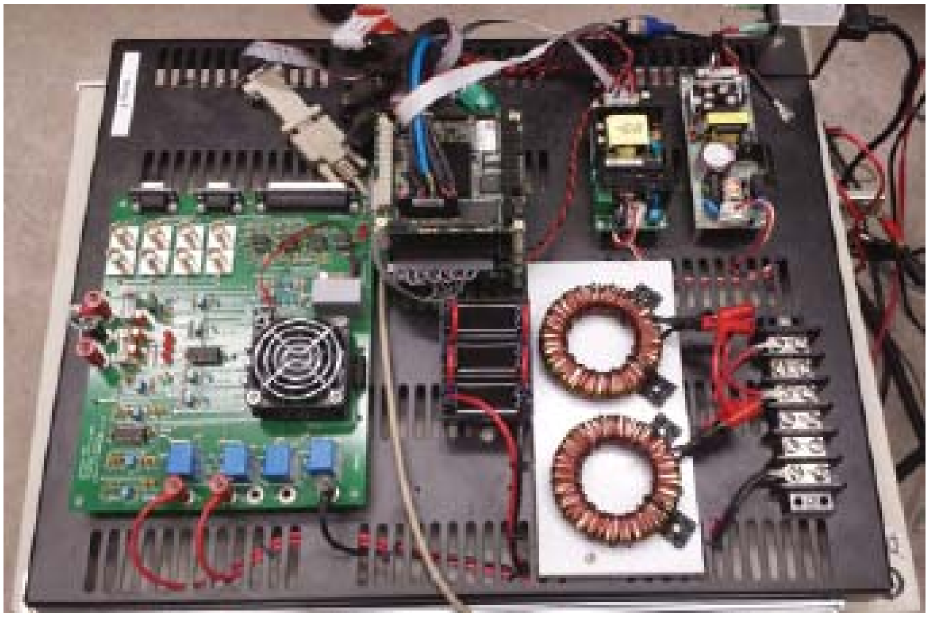


Figure 4.6: Power Electronic Building Block (PEBB)

concept gives a solution for integration of a large number of distributed generations without causing disruption in the utility network. Microgrids also allow for local control of the distributed generation units and attests.

Along with developing a robotic team capable of executing such missions, Michigan Technological University has also developed a microgrid testbed utilizing a robotic platform and a custom designed and built Power Electronic Building Block (PEBB) module (Fig. 4.6). This module is a medium to make adoption between power sources and power loads with different characteristics possible.

Each PEBB module is capable of power conversion of up to 1 kW at voltages up to 600 V. This hardware enables a very flexible and re-configurable platform to develop algorithms and controls for dc and ac microgrids. The PEBB design from Fig. 4.6 can be configured

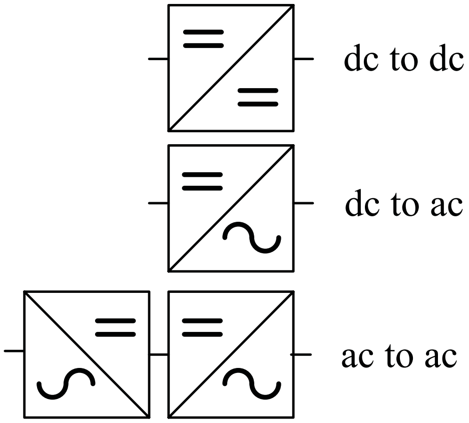


Figure 4.7: Types of conversion, dc-to-dc, dc-to-ac, and ac-to-ac through back-to-back dc to ac converters.

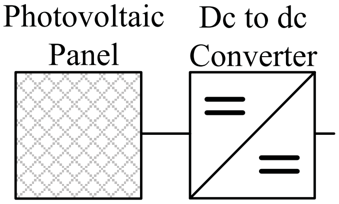


Figure 4.8: Photovoltaic panel and a regulated dc to dc converter with maximum power point tracking.

into several conversion processes. Fig. 4.7 shows three basic conversion processes and can be implemented by a common PEBB hardware architecture. The PEBB is mounted to the back of the robot and each of these conversion processes (dc-to-dc, dc-to-ac, and ac-to-ac) can be configured autonomously on-sight in response to the system needs.

In addition, other assets could be included. For example a PEBB can be pared with a solar panel array as shown in Fig. 4.8 to provide renewable energy to the system. Other assets, such as batteries or super capacitor banks could also be configured.

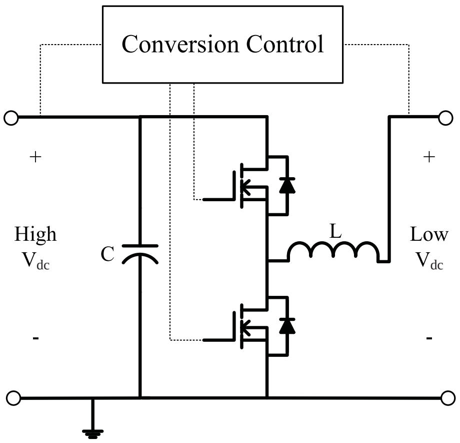

Conversion Control
+
High
V_{dc}
-
C
L
+
Low
V_{dc}
-

Figure 4.9: dc to dc conversion module

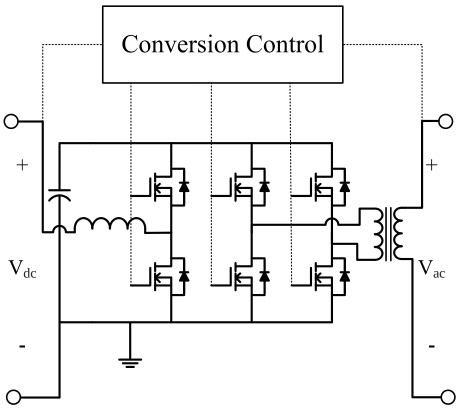

Conversion Control
+
V_{dc}
-
+
V_{ac}
-

Figure 4.10: dc to ac conversion module

The PEBB circuit shown in Fig. 4.6 can carry out several conversion processes. The basic dc-to-dc converter is shown in Fig. 4.9. This configuration converts energy from a high voltage dc to a low voltage dc and is a bi-directional conversion process, meaning that the current can flow in either direction by being controlled via the on-board control system. The PEBB circuit can also be configured to include a dc-to-ac conversion shown in Fig. 4.10 which also has bi-direction power flow.

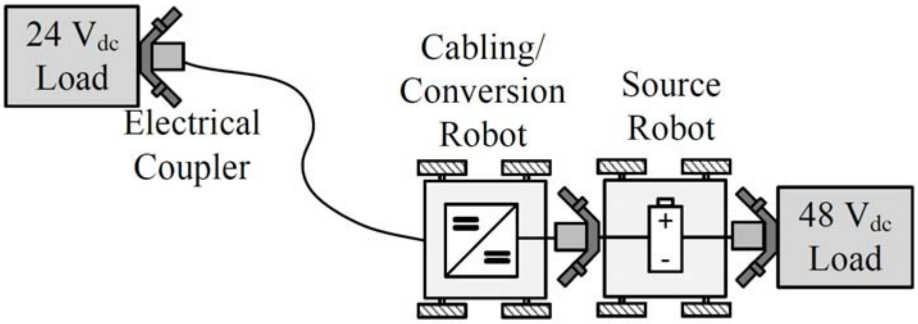


Figure 4.11: Combination of a power source robot and a cabling/conversion robot to power up a 48 V_{dc} and a 24 V_{dc} load

While robots are completing physical connection between each other or stationary nodes, they communicate their individual terminal voltage characteristics. Then each converter will configure their respective PEBBs to provide the necessary voltage and current. The mobile converter robot will configure the control system of the internal converter structure based on this information.

Fig. 4.11 shows a power source robot supplying a power load with 48 V_{dc} and a cabling/-conversion robot uses the same power source to supply a 24 V_{dc} power load using a PEBB which is the equivalent of Fig. 4.4 from power electrics point of view.

4.2.2 Ground Robot

For prototyping the algorithms, a small mobile robot platform known as DaNI (Fig. 4.5) was used. DaNI is part of the National Instruments Robotics Starter Kit and is made specifically for teaching and research purposes. The DaNI is equipped with the National Instruments sbRIO-9632. The National Instruments sbRIO (Single-Board Reconfigurable Input/Output) is an embedded controller designed with real-time processing and rapid prototyping in mind. It features a field-programmable gate array (FPGA) and I/O ports that allow communication with various sensors. The sbRIO allows network traffic by means of both web and file servers. If needed it can also support communication via RS-232 serial port. The 256MB on-board non volatile memory was utilized in order to: store and run the obstacle avoidance, path planning, and path tracking. For the close area obstacle recognition and distance measurement a simple IR sensor was used. This particular IR sensor has a range between 20 and 150cm. This sensor was mounted on a servo motor allowing a data return range between -75° to 75° in front of the DaNI.

4.2.3 Magnetic Electrical Connections

In this work, the goal is to make physical connection between a power source robot and two power load nodes. Establishing a good electrical connection is a major factor, along with

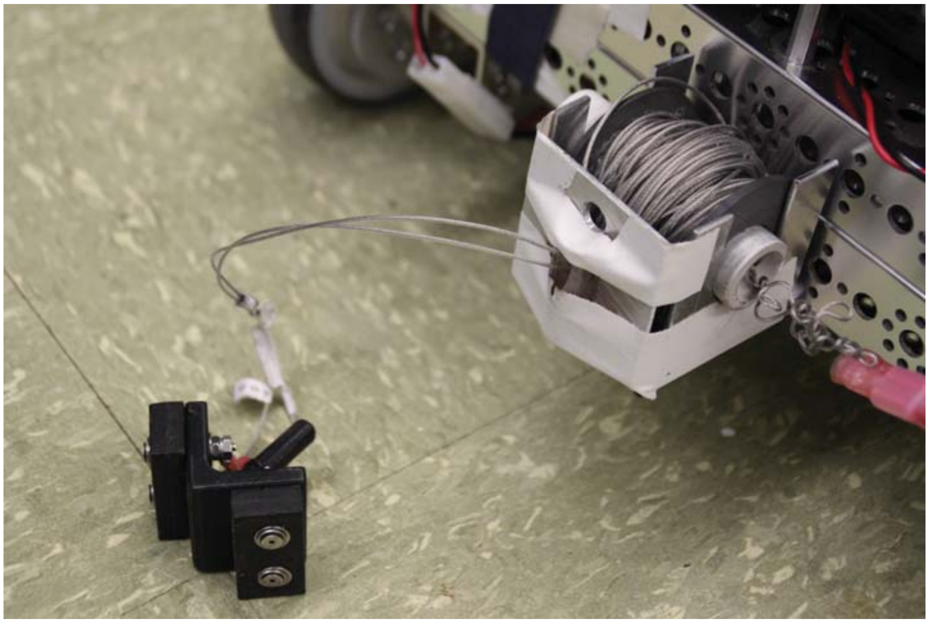


Figure 4.12: Male magnetic connector with wire pulley; Photo by author

performing in cluttered unknown areas. In this project a magnetic connection was designed to reduce the risk of faulty connections. This design is a reliable solution for establishing electrical connection without human intervention. For this design, neodymium magnet with a protective coating was selected. The magnets have two countersunk holes which facilitate mounting electrical connections through the magnet while remaining shielded of the magnet. The polarization property of the magnets allow the electrical connectors to align properly for every connection. The angled design of magnet mounts also helps with eliminating the risk of misalignment. Each connection includes a male (Fig. 4.12) and a female part (Fig. 4.13).

As the agent approaches the target, the male connector is absorbed by the female connector and pulls out of the mount on the agent and connects to female component. Two wires are

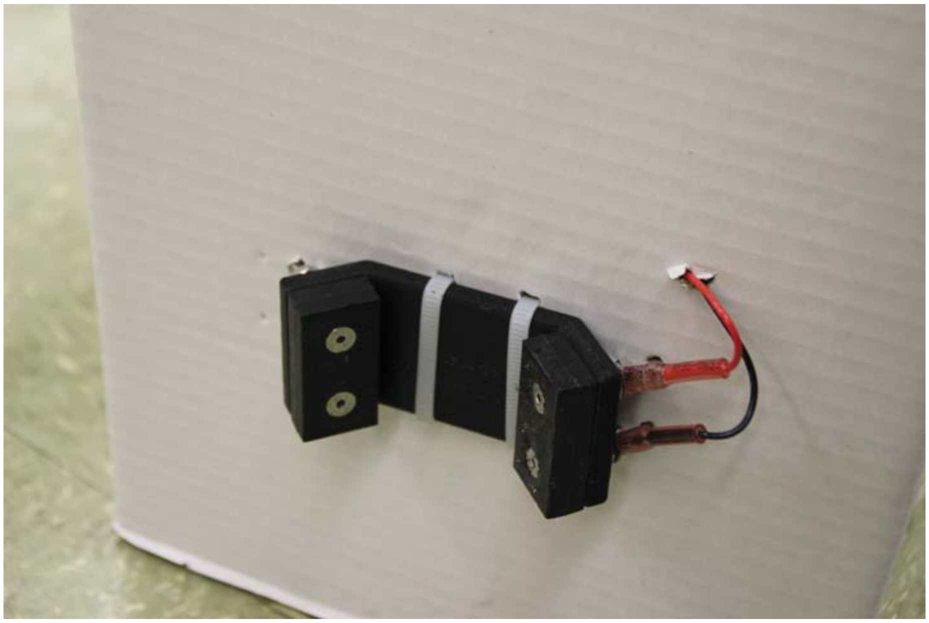


Figure 4.13: Female magnetic connector; Photo by author

connected to this male component that are rolled around a pulley (Fig. 4.12) and unwind as the agent moves away from the target and established electrical connection. The other side of these wires go through the pulley and come out of the center of it and connect to another pulley with the same structure with another male connector.

4.2.4 Camera System and IR Sensor

knowledge of position and state of targets, vehicle, and obstacles is the most important information needed for assuring a successful mission. This information is among the inputs of the hierarchy's blocks and is updated during the whole operation as a feedback to these blocks (see Fig. 3.1).

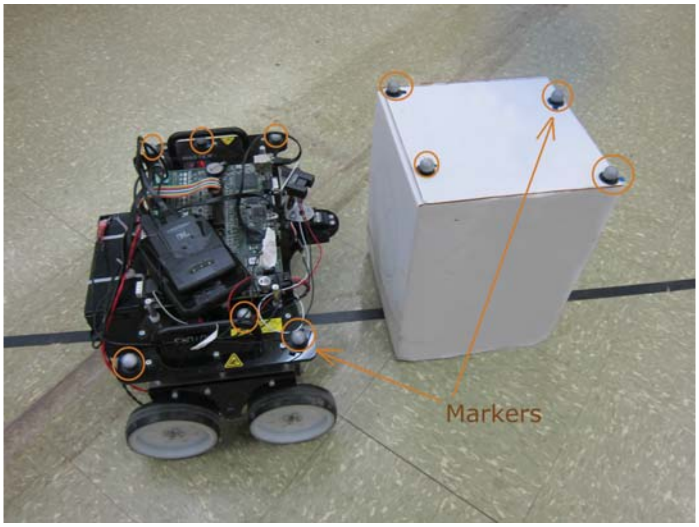


Figure 4.14: Marker placement on robot and objec; Photo by author

To accurately measure the position of the DaNIs and power loads, a Qualisys Oqus motion capture camera system is used. The camera system allows the detection and localization of objects. These cameras work by emitting light in the infrared (IR) spectrum and reflecting off of the IR markers (Fig. 4.14) within the area. These reflections are then recorded with a high frame rate. The camera software can track individual objects by means of the markers positioned on the actual object very accurately (Fig. 4.15). This approach of data gathering was used in order to imitate the GPS system.

A sharp GP2Y0A02YK0F Long Range Infrared Proximity Sensor was used to detect existence and distance of obstacles. This IR sensor has been mounted on a servo which oscillates to the right and left in order to imitate scans of a two dimensional laser range finder.

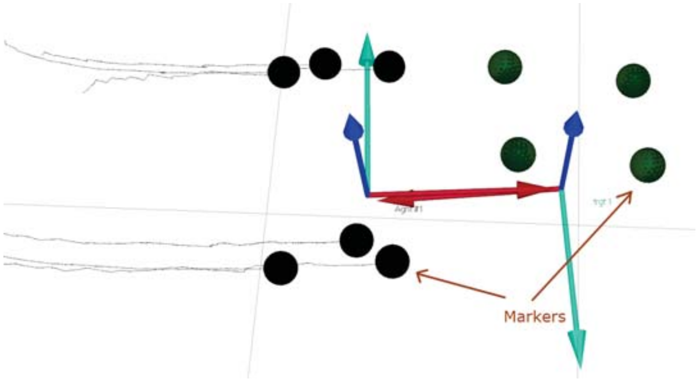


Figure 4.15: 3D representation of objects in workspace by camera system software utilizing markers

4.3 Results

The algorithms discussed in chapter 3 were validated in a lab-size setting, using hardware discussed in section 4.2. Before discussing the results of path planning algorithms, the performance of IR data filtering and directions choosing algorithm is analyzed. Failure in satisfactory performance of the IR sensor reading can have a drastic impact on the overall completion of the mission.

4.3.1 IR Filtering and Direction Choosing Algorithms Results for Simple Path Planning Algorithm

The key element for successful obstacle avoidance is having a robust performance from the combination of IR sensor and the algorithm which processes its data. Fig. 4.16 illustrates 1)

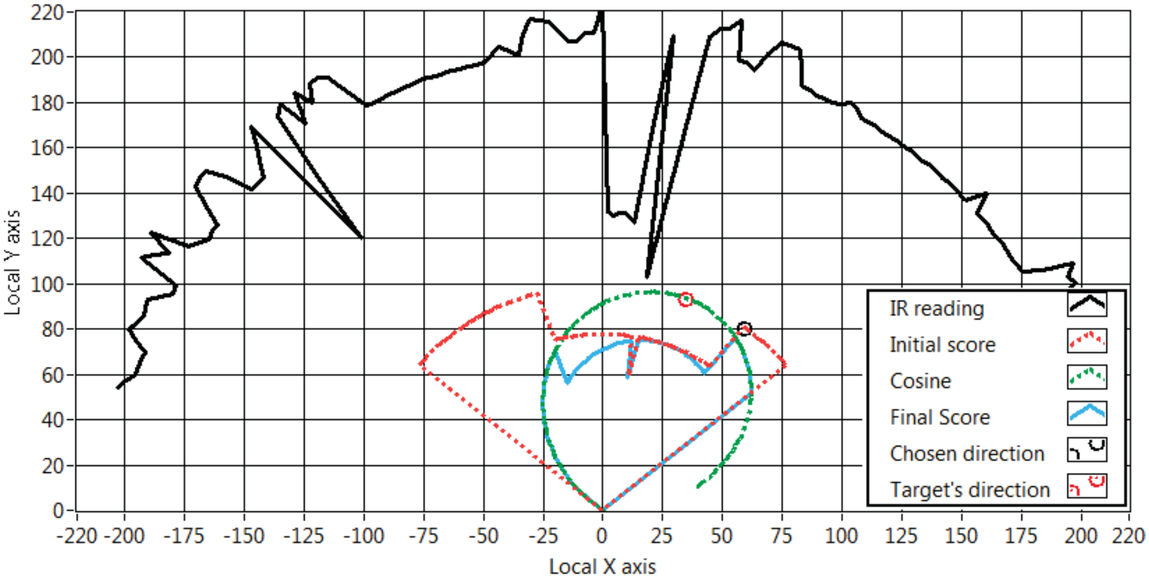


Figure 4.16: Example of IR readings, scoring, and direction choosing results

a sample of IR sensor readings, 2) process results of these readings based on the algorithm, and 3) the final output direction of direction choosing algorithm. The black line represents IR readings after going through a Butterworth filter. The red line shows the score of each angle multiplied by its neighbors. In order to have a better illustration of these scores, they are all multiplied by a factor of 100 before displaying them on the graph. The green line is cosine value of absolute of subtracting each angle from desired angle (target's direction). The blue line is the multiplication of scores represented by red line and cosine values. Finally the black circle shows the direction chosen for vehicle (by algorithm) in order to avoid obstacles.

As it can be seen the final direction produced by the IR algorithm (black circle) maintains a reasonable divergence from direction of obstacles which results in successful obstacle avoidance during the mission.

4.3.2 Simple Path Planning Algorithm Results

Satisfactory results from the process of IR reading leads us to analysis of path planing performance. The test setup for proof of concept includes two power loads, two autonomous ground vehicles, and a number of arbitrary obstacles with sufficient height for IR sensor reading. All of the mentioned components were positioned arbitrary (only requiring enough clearance in front of the target connections) and each robot is only given information addressing the position and orientation of power nodes (loads in this case) and itself through the camera system.

The path of each robot after successful completion of the mission is shown in Fig. 4.17. In this figure the blue line represent the first vehicle (power source agent) and its path, also the red line represent the second vehicle (cabling agent) and its path and the obstacles are shown by black circles.

Different stages of a completely successful mission can be seen in Fig. 4.17. Mission planner of power source agent guides the robot towards the imaginary point in front of the first power load and after reaching this point, the destination of path planner is changed to the actual touch point of this power load via the mission planner. By the time that first connection is made, the power source agent moves back from the electrical connection

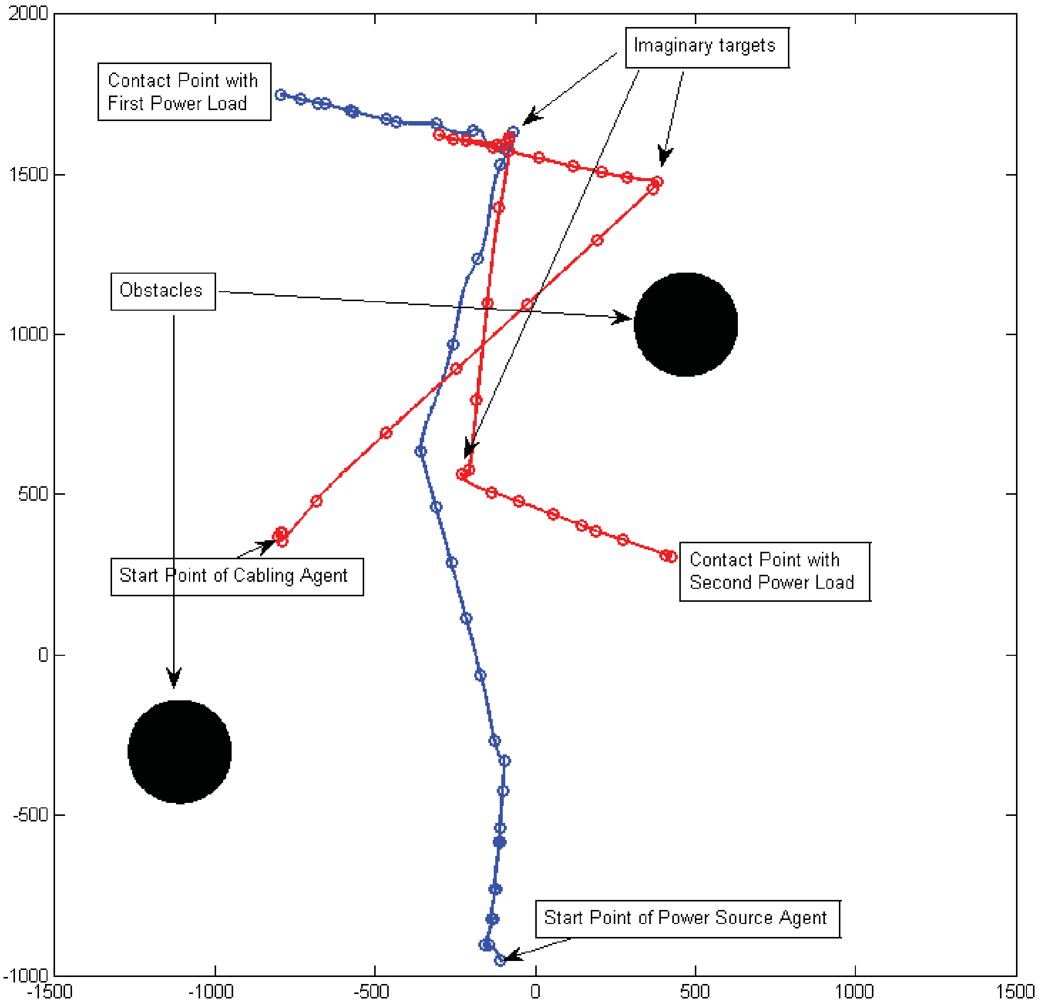


Figure 4.17: Test results of path planning of mission of two agents providing power to two power loads

touch point location due to safety reasons. At this point, the mission of power source agent is completed and the mission of cabling agent starts. First, robot starts moving towards an imaginary point behind the power source agent and then towards the touch point on the back of the power source robot. After making the connection and backing from the power source, the imaginary target in front of the second power load is set as the destination for cabling agent. The autonomous power distribution is finalized when the cabling agent makes the electrical connection with second power load. Fig. 4.17 illustrates that both agents managed to completely avoid any collision with obstacles while assembling power connection with both power loads.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

In this work, it was stated that a self-configuring autonomous microgrid has great impact on autonomous mission duration and providing power to other systems in need. The focus is using autonomous mobile power-grids to help re-establish power in areas with limited human resources. The organization, layout, and operation of such mobile robotic microgrids are highly dependent on the assets in the area of operation.

In this effort a lab-size hardware setup was used to demonstrate the capability of power distribution through physical wiring and efficient communication between the agents of the microgrid. A practical hierarchy was presented that accounts for uncertainty in predictions, minimal required information of the operation environment, minimal required communication between agents, and is scalable to real-size robots with different perception modules and sensor suites. Path planning is the main component of this hierarchy by providing a collision free path. For this purpose two algorithms were developed one for implementation on an unsophisticated lab platform as a proof-of-concept and a preliminary study on path planning requirements. The other algorithm was a geometric path planner addressing the limitations and assumptions derived from experimenting with simple path planner algorithm. For the power electric element of the project, a module being developed in Michigan Technological University was introduced that will be installed on the robotic platform to perform adaptation between power sources and power loads that are different in their electrical power format. This module equips robots with capability of changing the voltage and current type (ac versus dc) of the transferring power, that is provided either from a power source installed on the robot or through an external electric connection. For validation of autonomous power distribution concept a magnetic connection system was designed, prototyped, and manufactured to guarantee reliable physical electric connections between different nodes of the microgrid.

At the end, these components were implemented and tested in a lab environment and demonstrated a reliable autonomous power distribution system to navigate through obstacles, make electrical connections, and establish a microgrid. While the power module's development is in progress, the proof of concept was carried out with a power source robot

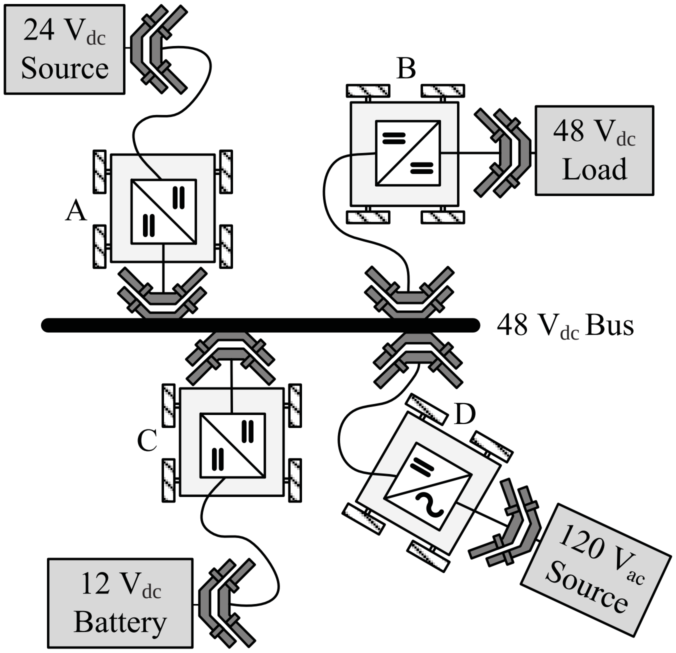


Figure 5.1: Four robot system. Robot A converts power from 24 V_{dc} source to 48 V_{dc} common bus. Robot B converts power from bus to 48 V_{dc} load. Robot C converts power to/from 12 V_{dc} battery. Robot D converts power to/from bus to 120 V_{ac} source.

equipped with a battery as source that is compatible with power loads characteristics and a cabling robot.

5.2 Future Developments

While we presented the results of this project up to this stage, there is much more being developed to the ultimate goal. The integration of autonomous mobile platforms and PEBB is the next step of this project to create self-organizing, ad-hoc microgrid. Below is a description of developments that will be added to the current system in near future.

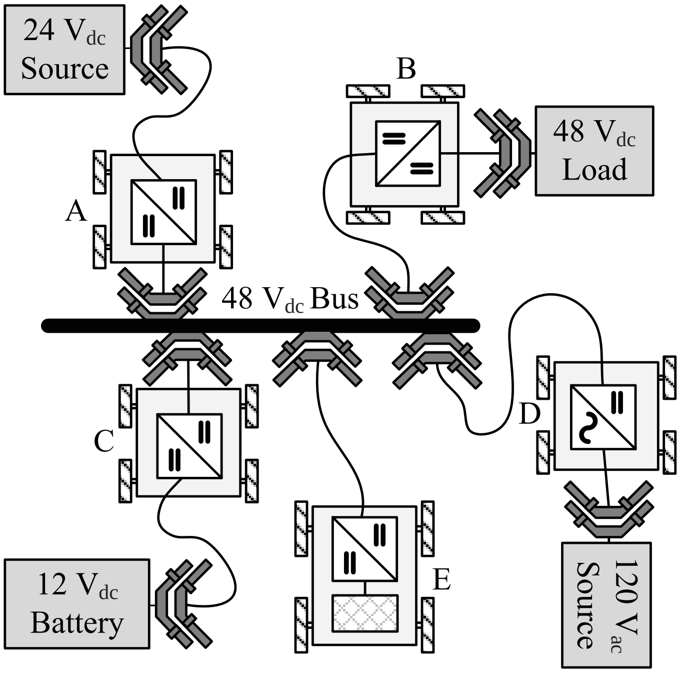


Figure 5.2: Four robot system. Robot A converts power from 24 V_{dc} source to 48 V_{dc} common bus. Robot B converts power from bus to 48 V_{dc} load. Robot C converts power to/from 12 V_{dc} battery. Robot D converts power to/from bus to 120 V_{ac} source. Robot E provides renewable energy from a on-board solar panel and regulated dc to dc converter.

Fig. 5.1 shows a larger system with four robot/PEBBs and uses a common 48 V_{dc} bus to create a microgrid with a 24 V_{dc} source, a 48 V_{dc} load, a 12 V_{dc} battery and a 120 V_{ac} power source. The robot/PEBBs communicate their positions and connection voltages to then configure the PEBBs to provide the proper voltages and currents through the common bus. The battery in Fig. 5.1 is used to support the system, charging and discharging depending upon the profile of the load.

Fig. 5.2 shows the system from Fig. 5.1 with the addition of a renewable energy robot/PEBB. In this configuration

We believe that once the physical electrical system is connected, intelligent power electronics on the robots will automatically determine the required energy conversion to meet the energy and power consumption of the loads. If loads, generation, or other assets change, then the nodes can physically and electrically reconfigure to meet the new demand or generate new configurations to compensate for renewable energy generation fluctuations and energy storage. Investigating optimal connection methods for establishing connections between microgrids is another area of focus. The connection positions will be optimized and in parallel reconfiguration mechanisms in presence of resource or power requirement changes will be determined. With this approach, the developed ad-hoc electric microgrid system will be highly reconfigurable and robust.

The robust path planning algorithm will be implemented on a real-size robotic platform in future for experimental validation. It will also be equipped with dynamic obstacle avoiding capability using a hierarchical fuzzy rule [61] to show satisfactory result in avoiding local minimas and deadlocks [62]. A new vehicle platform is under construction at this moment which will be equipped with an effective sensor suit and powerful embedded systems. The sensors utilized on this vehicle and the vehicle's dynamics have been modeled in the new simulation environment leading to better assessing the vehicle performance. This platform is a tracked vehicle with two electrical motors and a sensor suit consisted of a LIDAR (laser range finder), multiple infrared and sonar proximity sensors, an inertial measurement unit, a GPS, and two motion monitoring systems for each electrical motor. The control system of the vehicle is consisted of a xPC target machine for control algorithm and communicating with the LIDAR, a sbRIO 9642 for path tracking and communication with the rest of the sensors, and a RoboteQ motor controller to control the two electrical motor which actuate the tracks. This platform is capable of functioning in most of rough terrain settings and providing the potential for complete investigation of robustness of the control algorithm.

Considering environments with limited communication with GPS satellites such as indoors, accumulative errors of an inertial measurement unit, inaccurate output of magnetometers, lack of 3D perception of the surrounding, and the need for recognition of a target without information about the exact position justifies utilization of a vision sensor unit. The recent improvements in processing capabilities of modern embedded systems has opened a gate towards using real-time image processing algorithms in order to provide much more information to autonomous systems. In future developments vision sensors will be utilized for simultaneous localization and mapping (SLAM).

References

[1] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, *et al.*, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," *Journal of Field Robotics*, vol. 30, no. 1, pp. 44–63, 2013.

[2] W. H. Smith and K. M. Marks, "Seafloor in the malaysia airlines flight mh370 search area," *Eos, Transactions American Geophysical Union*, vol. 95, no. 21, pp. 173–174, 2014.

[3] H. Singh, J. G. Bellingham, F. Hover, S. Lemer, B. A. Moran, K. von der Heydt, and D. Yoerger, "Docking for an autonomous ocean sampling network," *Oceanic Engineering, IEEE Journal of*, vol. 26, no. 4, pp. 498–514, 2001.

[4] B. Mitchell, E. Wilkening, and N. Mahmoudian, "Low cost underwater gliders for littoral marine research," in *American Control Conference (ACC), 2013*, pp. 1412–1417, IEEE, 2013.

[5] F. P. Kemper, K. A. Suzuki, and J. R. Morrison, “Uav consumable replenishment: design concepts for automated service stations,” *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1-4, pp. 369–397, 2011.

[6] K. A. Suzuki, P. Kemper Filho, and J. R. Morrison, “Automatic battery replacement system for uavs: Analysis and design,” *Journal of Intelligent & Robotic Systems*, vol. 65, no. 1-4, pp. 563–586, 2012.

[7] A. Kwasinski, W. W. Weaver, P. L. Chapman, and P. T. Krein, “Telecommunications power plant damage assessment for hurricane katrina–site survey and follow-up results,” *IEEE Systems Journal*, vol. 3, no. 3, pp. 277–287, 2009.

[8] T. D. Ngo, H. Raposo, and H. Schioler, “Potentially distributable energy: Towards energy autonomy in large population of mobile robots,” in *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on*, pp. 206–211, IEEE, 2007.

[9] T. Deyle and M. Reynolds, “Surface based wireless power transmission and bidirectional communication for autonomous robot swarms,” in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 1036–1041, IEEE, 2008.

[10] C. Melhuish and M. Kubo, “Collective energy distribution: Maintaining the energy balance in distributed autonomous robots using trophallaxis,” in *Distributed Autonomous Robotic Systems 6*, pp. 275–284, Springer, 2007.

[11] A. Couture-Beil and R. T. Vaughan, “Adaptive mobile charging stations for multi-robot systems,” in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 1363–1368, IEEE, 2009.

[12] R. Cassinis, F. Tampalini, P. Bartolini, and R. Fedrigotti, “Docking and charging system for autonomous mobile robots,” *Department of Electronics for Automation, University of Brescia, Italy*, 2005.

[13] T. D. Ngo and H. Schiøler, “A truly autonomous robotic system through self-maintained energy,” in *Proceedings of the 23rd International Symposium on Automation and Robotics in Construction*, pp. 834–839, 2006.

[14] Y.-C. Wu, M.-C. Teng, and Y.-J. Tsai, “Robot docking station for automatic battery exchanging and charging,” in *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pp. 1043–1046, IEEE, 2009.

[15] R. H. Lasseter, “Microgrids,” in *IEEE Power Engineering Society Winter Meeting*, vol. 1, pp. 305–308, IEEE, 2002.

[16] K. A. Nigim and W. Lee, “Micro grid integration opportunities and challenges,” in *IEEE Power Engineering Society General Meeting*, pp. 1–6, IEEE, 2007.

[17] W. W. Weaver, N. Mahmoudian, and G. Parker, “Autonomous mobile power blocks for prepositioned power conversion and distribution,” in *TARDEC Ground Vehicle Systems Engineering and Technology Symposium*, 2012.

[18] M. Erol-Kantarci, B. Kantarci, and H. Mouftah, “Reliable overlay topology design for the smart microgrid network,” *IEEE Network*, vol. 25, no. 5, pp. 38–43, 2011.

[19] G. Liu and J. C. Trinkle, “Complete path planning for planar closed chains among point obstacles.,” in *Robotics: Science and Systems*, vol. 170, 2005.

[20] D. Zhu and J.-C. Latombe, “New heuristic algorithms for efficient hierarchical path planning,” *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 1, pp. 9–20, 1991.

[21] N. I. Katevas, S. G. Tzafestas, and C. G. Pnevmatikatos, “The approximate cell decomposition with local node refinement global path planning method: Path nodes refinement and curve parametric interpolation,” *Journal of Intelligent and Robotic Systems*, vol. 22, no. 3-4, pp. 289–314, 1998.

[22] L. Zhang, Y. J. Kim, and D. Manocha, “Efficient cell labelling and path non-existence computation using c-obstacle query,” *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1246–1257, 2008.

[23] I. Kamon and E. Rivlin, “Sensory-based motion planning with global proofs,” *Robotics and Automation, IEEE Transactions on*, vol. 13, no. 6, pp. 814–822, 1997.

[24] V. J. Lumelsky and A. A. Stepanov, “Dynamic path planning for a mobile automaton with limited information on the environment,” *Automatic Control, IEEE Transactions on*, vol. 31, no. 11, pp. 1058–1063, 1986.

[25] I. Kamon, E. Rivlin, and E. Rimon, “A new range-sensor based globally convergent navigation algorithm for mobile robots,” in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1, pp. 429–435, IEEE, 1996.

[26] J. Borenstein and Y. Koren, “The vector field histogram-fast obstacle avoidance for mobile robots,” *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 278–288, 1991.

[27] I. Ulrich and J. Borenstein, “Vfh+: Reliable obstacle avoidance for fast mobile robots,” in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 2, pp. 1572–1577, IEEE, 1998.

[28] I. Ulrich and J. Borenstein, “Vfhˆ*: Local obstacle avoidance with look-ahead verification,” in *ICRA*, pp. 2505–2511, 2000.

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

[30] M. Seder and I. Petrovic, “Dynamic window based approach to mobile robot motion control in the presence of moving obstacles.,” in *ICRA*, vol. 7, pp. 1986–1991, 2007.

[31] Z. Shiller and Y.-R. Gwo, “Dynamic motion planning of autonomous vehicles,” *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 2, pp. 241–249, 1991.

[32] O. Brock and O. Khatib, “High-speed navigation using the global dynamic window approach,” in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1, pp. 341–346, IEEE, 1999.

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

[34] T. Sattel and T. Brandt, “Ground vehicle guidance along collision-free trajectories using elastic bands,” in *American Control Conference, 2005. Proceedings of the 2005*, pp. 4991–4996, IEEE, 2005.

[35] S. K. Gehrig and F. J. Stein, “Elastic bands to enhance vehicle following,” in *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pp. 597–602, IEEE, 2001.

[36] V. Delsart and T. Fraichard, “Navigating dynamic environments using trajectory deformation,” in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 226–233, IEEE, 2008.

[37] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[38] C. W. Warren, “Multiple robot path coordination using artificial potential fields,” in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pp. 500–505, IEEE, 1990.

[39] E. G. Hernández-Martínez and E. Aranda-Bricaire, "Non-collision conditions in multi-agent robots formation using local potential functions," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 3776–3781, IEEE, 2008.

[40] O. Brock and O. Adviser-Khatib, *Generating robot motion: The integration of planning and execution.* Stanford University, 2000.

[41] S. Kakos and K. J. Kyriakopoulos, "The navigation functions approach for the label anti-overlapping problem," in *Proceedings of the 4th eurocontrol innovative research workshop, Paris, France*, 2005.

[42] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," *IEEE Transactions on robotics and automation*, vol. 16, no. 5, pp. 615–620, 2000.

[43] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous Robots*, vol. 13, no. 3, pp. 207–222, 2002.

[44] L. Yin and Y. Yin, "An improved potential field method for mobile robot path planning in dynamic environments," in *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pp. 4847–4852, IEEE, 2008.

[45] L. Huang, "Velocity planning for a mobile robot to track a moving targetâĂŤa potential field approach," *Robotics and Autonomous Systems*, vol. 57, no. 1, pp. 55–63, 2009.

[46] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial intelligence*, vol. 114, no. 1, pp. 3–55, 1999.

[47] A. Sannino, G. Postiglione, and M. H. J. Bollen, "Feasibility of a dc network for commercial facilities," *IEEE Transactions on Industry Applications*, vol. 39, no. 5, pp. 1499–1507, 2003.

[48] D. Salomonsson and A. Sannino, "Low-voltage dc distribution system for commercial power systems with sensitive electronic loads," *IEEE Transactions on Power Delivery*, vol. 22, no. 3, pp. 1620–1627, 2007.

[49] J. M. Guerrero, J. C. Vasquez, J. Matas, L. G. de Vicuña, and M. Castilla, "Hierarchical control of droop-controlled ac and dc microgridsâĂŤa general approach toward standardization," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 1, pp. 158–172, 2011.

[50] X. Wang, J. M. Guerrero, F. Blaabjerg, and Z. Chen, "A review of power electronics based microgrids," *Journal of Power Electronics*, vol. 12, no. 1, pp. 181–192, 2012.

[51] F. Arrichiello, S. Chiaverini, and V. K. Mehta, "Experiments of obstacles and collision avoidance with a distributed multi-robot system," in *Information and Automation (ICIA), 2012 International Conference on*, pp. 727–732, IEEE, 2012.

[52] S. Bhattacharya, M. Likhachev, and V. Kumar, “Multi-agent path planning with multiple tasks and distance constraints,” in *IEEE International Conference on Robotics and Automation (ICRA), 2010*, pp. 953–959, 2010.

[53] S. Liu, D. Sun, and C. Zhu, “Coordinated motion planning for multiple mobile robots along designed paths with formation requirement,” *IEEE/ASME Transactions on Mechatronics Journal*, vol. 16, no. 6, pp. 1021–1031, 2011.

[54] P. Raja and S. Pugazhenthi, “Optimal path planning of mobile robots: A review,” *International Journal of Physical Sciences*, vol. 7, no. 9, pp. 1314–1320, 2012.

[55] A. Reina, C. Pinciroli, E. Ferrante, A. E. Turgut, R. OâĂŹGrady, M. Birattari, and M. Dorigo, “Closed-loop aerial robot-assisted navigation of a cohesive ground-based robot swarm,” 2011.

[56] G. Parker, W. W. Weaver, and S. Goldsmith, “Integrated, interconnected and intelligent microgrids,” in *Advanced migrogrid concepts and technologies workshop*, 2012.

[57] C. E. Ekneligoda and W. W. Weaver, “Optimal team communication structures in microgrids,” in *IEEE Energy Conversion Congress and Exposition (ECCE)*, pp. 943–949, 2011.

[58] R. R. Murphy, “A decade of rescue robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012*, pp. 5448–5449, 2012.

[59] R. R. Murphy, K. L. Dreger, S. Newsome, J. Rodocker, B. Slaughter, R. Smith, E. Steimle, T. Kimura, K. Makabe, K. Kon, *et al.*, "Marine heterogeneous multirobot systems at the great eastern japan tsunami recovery," *Journal of Field Robotics*, vol. 29, no. 5, pp. 819–831, 2012.

[60] S. Kawatsuma, M. Fukushima, and T. Okada, "Emergency response by robots to fukushima-daiichi accident: summary and lessons learned," *Industrial Robot: An International Journal*, vol. 39, no. 5, pp. 428–435, 2012.

[61] T. Aoki, M. Matsuno, T. Suzuki, and S. Okuma, "Motion planning for multiple obstacles avoidance of autonomous mobile robot using hierarchical fuzzy rules," in *Multisensor Fusion and Integration for Intelligent Systems, 1994. IEEE International Conference on MFI'94.*, pp. 265–271, IEEE, 1994.

[62] C. Lin and L. Wang, "Intelligent collision avoidance by fuzzy logic control," *Robotics and Autonomous Systems*, vol. 20, no. 1, pp. 61–83, 1997.

[63] G. Klancar, D. Matko, and S. Blazic, "Mobile robot control on a reference path," in *Proceedings of the 2005 IEEE International Symposium on Intelligent Control, 2005. Mediterrean Conference on Control and Automation*, pp. 1343–1348, IEEE, 2005.

[64] D. Stormont, "Autonomous rescue robot swarms for first responders," in *IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*, 2005.

[65] M. Hsieh, A. Cowley, V. Kumar, and C. Taylor, “Towards the deployment of a mobile robot network with end-to-end performance guarantees,” in *IEEE International Conference on Robotics and Automation(ICRA)*, 2006.

[66] E. Budianto, A. Hafidh, F. Al Afif, A. Wibowo, W. Jatmiko, B. Hardian, P. Mursanto, and A. Muis, “Autonomous telecommunication networks coverage area expansion in disaster area using mobile robots,” in *International Symposium on Micro-NanoMechatronics and Human Science (MHS)*, 2011.

[67] U. Witkowski, M. El-Habbal, S. Herbrechtsmeier, A. Tanoto, J. Penders, L. Alboul, and V. Gazi, “Ad-hoc network communication infrastructure for multi-robot systems in disaster scenarios,” in *the EURON/IARP International Workshop on Robotics for Risky Interventions and Surveillance of the Environment*, 2008.

[68] R. S. J. Sparks, W. P. Aspinall, N. A. Chapman, B. E. Hill, D. J. Kerridge, J. Pooley, and C. A. Taylor, “Catastrophic impacts of natural hazards on technological facilities and infrastructure,” *Risk and Uncertainty Assessment for Natural Hazards Journal*, p. 445, 2013.

[69] M. F. Habib, M. Tornatore, F. Dikbiyik, and B. Mukherjee, “Disaster survivability in optical communication networks,” *Computer Communications Journal*, 2013.

[70] D. Wilson, R. Robinett, and S. Goldsmith, “Renewable energy microgrid control with energy storage integration,” in *International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, pp. 158–163, 2012.

[71] R. D. Robinett, D. G. Wilson, and S. Y. Goldsmith, “Collective control of networked microgrids with high penetration of variable resources part i: Theory,” in *IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems*, pp. 1–4, 2012.

[72] M. C. Silverman, D. Nies, B. Jung, and G. Sukhatme, “Staying alive: A docking station for autonomous robot recharging,” in *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*, vol. 1, pp. 1050–1055, IEEE, 2002.

[73] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans, “Modular reconfigurable robots in space applications,” *Autonomous Robots*, vol. 14, no. 2-3, pp. 225–237, 2003.

[74] W. M. DeBusk, “Unmanned aerial vehicle systems for disaster relief: Tornado alley,” in *AIAA Infotech@ Aerospace Conference, AIAA-2010-3506, Atlanta, GA*, 2010.

[75] T. McGinnis, C. P. Henze, and K. Conroy, “Inductive power system for autonomous underwater vehicles,” in *OCEANS 2007*, pp. 1–5, IEEE, 2007.

[76] L. A. Gish, *Design of an AUV recharging system.* PhD thesis, Monterey California. Naval Postgraduate School, 2004.

[77] M. Maasoumy, B. Moridian, M. Razmara, M. Shahbakhti, and A. Sangiovanni-Vincentelli, “Online simultaneous state estimation and parameter adaptation for

building predictive control," in *ASME 2013 Dynamic Systems and Control Conference*, pp. V002T23A006–V002T23A006, American Society of Mechanical Engineers, 2013.

[78] E. M. Ficanha, M. Rastgaar, B. Moridian, and N. Mahmoudian, "Ankle angles during step turn and straight walk: Implications for the design of a steerable ankle-foot prosthetic robot," in *ASME 2013 Dynamic Systems and Control Conference*, pp. V001T09A001–V001T09A001, American Society of Mechanical Engineers, 2013.

[79] M. Barzin, B. Daryl, M. Nina, W. Wayn, and R. Rush, "Autonomous power distribution system," in *International Federation of Automatic Control. World Congress (19th)*, International Federation of Automatic Control, 2014.

Appendix A

Navigation Code for Unsophisticated Platform

Appendix B

Robust Path Planning Algorithm Codes

B.1 Front End

```
clc

close all

clear all

t=0;

SimLength=3;

TimeStep=.01;

prompt = 'Wanna Record Results? ';

record = input(prompt,'s');
```

```
display(record);
recordBool=strcmp(record,'Yes')||strcmp(record,'yes')||
   ↪ strcmp(record,'y')||strcmp(record,'Y');
if recordBool
    c=fix(clock);
    cstr=strcat(mat2str(c),'.avi');
    SimulationVideo = VideoWriter(cstr);
    open(SimulationVideo);
    axis tight
    set(gca,'nextplot','replacechildren');
    set(gcf,'Renderer','zbuffer');

end
XRecord=nan*ones(1,SimLength/TimeStep+1);
YRecord=nan*ones(1,SimLength/TimeStep+1);
Base=SimBase();
ImportMap(Base,'workspace.bmp');
agent=RobotStates();
hold on
while t<SimLength&&abs((agent.X-agent.TargetX)^2+(agent.Y-
   ↪ agent.TargetY)^2)>100
```

```
    clc
    Show(agent);
    ShowField(agent);
    MoveRobot(agent,Base,TimeStep);
    drawnow
    t=t+TimeStep;
    if recordBool
        frame = getframe;
        writeVideo(SimulationVideo,frame);

    end
    XRecord(round(t/TimeStep+1))=[agent.X];
    YRecord(round(t/TimeStep+1))=[agent.Y];
    display(t);
end
plot(XRecord,YRecord,'c','LineWidth',1.5)
if recordBool
    close(SimulationVideo);
end
```

B.2 Workspace Object Definition

```
classdef SimBase < handle

    properties

        Width=400;

        Length=800;

        Matrix;


    end




    methods

        function Base=SimBase(Width,Length)

            if(nargin>0)

                Base.Width=Width;

                Base.Length=Length;

            end

            Base.Matrix=zeros(Base.Width,Base.Length);

        end
```

```
     function Obj=ImportMap(Obj,str)
         Obj.Matrix=flipud(imread(str));
         Obj.Width=size(Obj.Matrix,1);
         Obj.Length=size(Obj.Matrix,2);
         Show(Obj);
     end


     function Base=AddWall(Base,StartPoint,Width,Length)
         Base.Matrix(StartPoint(2):(StartPoint(2)+Length)
↪  ,StartPoint(1):(StartPoint(1)+Width))=1;
     end


     function Show(Base)
         BaseFigure = figure;
         set(BaseFigure,’name’,’Figure of the workspace’,
↪  ’numbertitle’,’off’)
         imshow(Base.Matrix);
         set(gca, ’Ydir’, ’normal’);
```

```
            set(gca, 'XLim', [1 Base.Length], 'YLim', [1
  ↪ Base.Width]);
        end
    end



end
```

B.3 Robot Object Definition

```
classdef RobotStates < handle & Lidar
    properties
        Name;
        X=150;
        Y=120;
        Theta=0;
        Length=100;
        Width=50;
        LidarOffCenter=40;
        TangentVelocity=0;
        AngularVelocity=0;
        MaxTangentVelocity=200;
        SafetyRadius;
        Rank=1;
        FieldNumber=5;
        TargetX=1200;
        TargetY=400;
        RobotFigures;
        FieldPlot=[];
        ArrowsPlot=[8,8,8];
        Mass=1;
        FieldDampDistMemory;
        EdgePlot;
    end
    methods
        function obj=RobotStates()
            obj=obj@Lidar();
            obj.FieldDampDistMemory=zeros(obj.AngularRange/
  ↪ obj.ScanStepSize+1,1);
            %if(nargin==1)
            %end
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Show
    function Show(obj)
        D=20;
        if ishandle(obj.RobotFigures)
            delete(obj.RobotFigures);
        end
        [PatchX,PatchY]=RobotCorners(obj);
        obj.RobotFigures(1)=line([PatchX,PatchX(1)],[
↪ PatchY,PatchY(1)],'Color','w');       %shows robot
        XLidar=obj.X+cos(obj.Theta*pi/180)*obj.
↪ LidarOffCenter;
        YLidar=obj.Y+sin(obj.Theta*pi/180)*obj.
↪ LidarOffCenter;
        obj.RobotFigures(2)=rectangle('Position',[XLidar
↪ -D/2,YLidar-D/2,D,D],...    %shows lidar
            'Curvature',[1,1],...
            'LineWidth',1.5,'edgecolor','r');
        obj.RobotFigures(3)=line([XLidar,XLidar+D*cos(
↪ obj.Theta*pi/180)/2],[YLidar,YLidar+D*sin(obj.Theta*pi
↪ /180)/2],'Color','r','LineWidth',3);    %shows rest of
↪  lidar
        daspect([1,1,1]);
        obj.RobotFigures(4)=rectangle('Position',[obj.
↪ TargetX-10,obj.TargetY-10,20,20],...    %shows target
            'Curvature',[1,1],...
            'LineWidth',1.5,'edgecolor','c');
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Robot Corners
    function [Xs,Ys]=RobotCorners(obj)
        Xs=[...     %Robot corners
            obj.X+cos(obj.Theta*pi/180)*(obj.Length/2)+
↪ sin(obj.Theta*pi/180)*obj.Width/2,...
            obj.X+cos(obj.Theta*pi/180)*(obj.Length/2)-
↪ sin(obj.Theta*pi/180)*obj.Width/2,...
            obj.X-cos(obj.Theta*pi/180)*(obj.Length/2)-
↪ sin(obj.Theta*pi/180)*obj.Width/2,...
```

```
            obj.X-cos(obj.Theta*pi/180)*(obj.Length/2)+
↪ sin(obj.Theta*pi/180)*obj.Width/2];
        Ys=[...
            obj.Y+sin(obj.Theta*pi/180)*(obj.Length/2)-
↪ cos(obj.Theta*pi/180)*(obj.Width/2),...
            obj.Y+sin(obj.Theta*pi/180)*(obj.Length/2)+
↪ cos(obj.Theta*pi/180)*(obj.Width/2),...
            obj.Y-sin(obj.Theta*pi/180)*(obj.Length/2)+
↪ cos(obj.Theta*pi/180)*(obj.Width/2),...
            obj.Y-sin(obj.Theta*pi/180)*(obj.Length/2)-
↪ cos(obj.Theta*pi/180)*(obj.Width/2)];
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Show Field
    function ShowField(obj)
        if ishandle(obj.FieldPlot)
            delete(obj.FieldPlot)
        end
        obj.SafetyRadius=.6*sqrt(obj.Length^2+obj.Width
↪ ^2);
        k=3;
```

```
            C=2;
            Er=abs(obj.TangentVelocity/(C*obj.
   ↪ MaxTangentVelocity));
            PlotTheta=(0:360)*pi/180;
            temp=1;
            for e=0:Er/(obj.FieldNumber-1):Er
                PlotDist=(k*e*obj.SafetyRadius*obj.Rank./(1-
   ↪ e*cos(PlotTheta))+obj.SafetyRadius);
%                 obj.FieldPlot(temp)=plot(PlotDist.*cos(
   ↪ PlotTheta+obj.Theta*pi/180)+obj.X, PlotDist.*sin(
   ↪ PlotTheta+obj.Theta*pi/180)+obj.Y,'w');
                temp=temp+1;
            end
            clear temp;
            daspect([1,1,1]);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %% Lidar Data
        function [LidarReadingAngles,LidarReadingDistances]=
   ↪ LidarData(obj,Base)
```

```
        [LidarReadingAngles,LidarReadingDistances]=
↪ LidarData@Lidar(obj,Base);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Translate to robot refrence
    function [DistancesVectorRobotRefrence,
↪ ThetasVectorRobotRefrence]=LidarToRobotCenter(
↪ DistancesVectorLidarReference,
↪ ThetasVectorLidarReference,obj)
        DistancesVectorRobotRefrenceX=
↪ DistancesVectorLidarReference.*cos(
↪ ThetasVectorLidarReference*pi/180)+obj.LidarOffCenter;
        DistancesVectorRobotRefrenceY=
↪ DistancesVectorLidarReference.*sin(
↪ ThetasVectorLidarReference*pi/180);
        DistancesVectorRobotRefrence=sqrt(
↪ DistancesVectorRobotRefrenceX.^2+
↪ DistancesVectorRobotRefrenceY.^2);
        ThetasVectorRobotRefrence=atan2(
↪ DistancesVectorRobotRefrenceY,
↪ DistancesVectorRobotRefrenceX)*180/pi;
```

```
     end
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     %% Force of each reading
     function ForceMag=ForceArrayOfReading(
↪ DistancesVectorRobotRefrence,ThetasVectorRobotRefrence
↪ ,obj,TimeStep)
         NumberOfReadings=length(
↪ DistancesVectorRobotRefrence);
         obj.SafetyRadius=.6*sqrt(obj.Length^2+obj.Width
↪ ^2);
         DistancesVectorRobotRefrence=
↪ DistancesVectorRobotRefrence-obj.SafetyRadius;
         k=3;
         C=2;
         rho0=.1;
         ForceMax=0;
         ForceMag=zeros(NumberOfReadings,1);
         Er=abs(obj.TangentVelocity/(C*obj.
↪ MaxTangentVelocity));
         Dmax=k*Er*obj.SafetyRadius*obj.Rank./(1-Er*cos(
↪ ThetasVectorRobotRefrence));
```

```
        Dmin=rho0*Dmax;
        DampRatio=0;
        for i=1:NumberOfReadings
            if Dmax(i)<10
                Dmax(i)=10;
            end
            if DistancesVectorRobotRefrence(i)>=Dmax(i)
                ForceMag(i)=0;
            else
                ForceMag(i)=(Dmax(i)-
↪ DistancesVectorRobotRefrence(i))*ForceMax/(Dmax(i)-
↪ Dmin(i))...
                    -DampRatio*(
↪ DistancesVectorRobotRefrence(i)-obj.
↪ FieldDampDistMemory(i))/TimeStep;
            end
        end
        obj.FieldDampDistMemory=
↪ DistancesVectorRobotRefrence;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
     %% Repulsion force calculation
     function [RepulsionForceXTotal,RepulsionForceYTotal
↪ ]=RepulsionForce(ForceMag,ThetasVectorRobotRefrence,
↪ obj)
         RepulsionForceX=cos(ThetasVectorRobotRefrence*pi
↪ /180).*ForceMag;
         RepulsionForceY=sin(ThetasVectorRobotRefrence*pi
↪ /180).*ForceMag;
         RepulsionForceXTotal=sum(RepulsionForceX);
         RepulsionForceYTotal=sum(RepulsionForceY);
         RepulsionForceMagTotal=sqrt(RepulsionForceYTotal
↪ ^2+RepulsionForceXTotal^2);
         RepulsionForceAngleTotal=atan2(
↪ RepulsionForceYTotal,RepulsionForceXTotal)+pi;
     end
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     %% Forces
     function [ForcesSummationMagnitude,
↪ ForcesSummationAngle]=...
```

```
            Forces(DistancesVectorLidarReference,
↪ ThetasVectorLidarReference,obj,TimeStep,
↪ AttractionForceAngle)
        % Repulsion
        [DistancesVectorRobotRefrence,
↪ ThetasVectorRobotRefrence]=LidarToRobotCenter(
↪ DistancesVectorLidarReference,
↪ ThetasVectorLidarReference,obj);
        ForceMag=ForceArrayOfReading(
↪ DistancesVectorRobotRefrence,ThetasVectorRobotRefrence
↪ ,obj,TimeStep);
        for i=1:length(obj.ArrowsPlot)
            if ishandle(obj.ArrowsPlot(i))
                delete(obj.ArrowsPlot(i));
            end
        end
        [RepulsionForceXTotal,RepulsionForceYTotal]=
↪ RepulsionForce(ForceMag,ThetasVectorRobotRefrence,obj)
↪ ;
        % attraction force
        Q=80;
```

```
         % Summation
         ForcesSummationX=-RepulsionForceXTotal+Q⋆cos(
↪ AttractionForceAngle);
         ForcesSummationY=-RepulsionForceYTotal+Q⋆sin(
↪ AttractionForceAngle);
         ForcesSummationMagnitude=sqrt(ForcesSummationX
↪ ^2+ForcesSummationY^2);
         if ForcesSummationMagnitude<30
             ForcesSummationMagnitude=30;
         end
         ForcesSummationAngle=atan2(ForcesSummationY,
↪ ForcesSummationX);
         %             obj.ArrowsPlot(3)=arrow([obj.X obj
↪ .Y],[obj.X+ForcesSummationMagnitude⋆cos(
↪ ForcesSummationAngle),...
         %                 obj.Y+ForcesSummationMagnitude
↪ ⋆sin(ForcesSummationAngle)],...
         %                 ’EdgeColor’,’m’, ’FaceColor’,’
↪ m’);
     end
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    %% Potential Field
    function [ForcesSummationMagnitude,
↪ ForcesSummationAngle]=PotentialFieldController(obj,
↪ Base,TimeStep)
        [LidarReadingAngles,LidarReadingDistances]=
↪ LidarData(obj,Base);
        AttractionForceAngle=GapFinding(
↪ LidarReadingAngles,LidarReadingDistances,obj);
        [ForcesSummationMagnitude,ForcesSummationAngle
↪ ]=...
            Forces(LidarReadingDistances,
↪ LidarReadingAngles,obj,TimeStep,AttractionForceAngle);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Obstacle Recognition
    function index=obstacleRecognition(obj,
↪ DistancesVectorLidarReference,
↪ ThetasVectorLidarReference)
        readings=length(DistancesVectorLidarReference);
        index=nan*ones(readings,1);
        noOfReturns=1;
```

```
         range=.95*obj.DistanceRange;
         for limitFinder=1:readings
             if DistancesVectorLidarReference(limitFinder
↪ )>range
                 DistancesVectorLidarReference(
↪ limitFinder)=nan;
                 ThetasVectorLidarReference(limitFinder)=
↪ nan;
             else
                 index(limitFinder)=noOfReturns;
                 noOfReturns=noOfReturns+1;
             end %if
         end %for
         DistancesVectorLidarReference(isnan(
↪ DistancesVectorLidarReference))=[];
         ThetasVectorLidarReference(isnan(
↪ ThetasVectorLidarReference))=[];
         if ~isempty(ThetasVectorLidarReference)
             [readingsX,readingsY]=pol2cart(
↪ ThetasVectorLidarReference*pi/180,
↪ DistancesVectorLidarReference);
```

```
            firstObstacleEnd=1;
            for obstacleEndIteration=1:int32(obj.
↪ AngularRange/obj.ScanStepSize)+1
                for limitFinder=length(readingsX):-1:
↪ obstacleEndIteration
                    if ((readingsX(limitFinder)-
↪ readingsX(obstacleEndIteration))^2+(readingsY(
↪ limitFinder)-readingsY(obstacleEndIteration))^2)<=4*
↪ obj.SafetyRadius^2;
                        firstObstacleEnd=limitFinder;
                        break
                    end%if
                end %for
                if firstObstacleEnd==
↪ obstacleEndIteration
                    break % end found
                end
            end
            numberOfObstacles=0;
            scanStart=firstObstacleEnd+1;
            obstacleElement=firstObstacleEnd+1;
            obstacleLimits=[nan,nan];
            while obstacleLimits(end)~=mod(
↪ firstObstacleEnd-1,length(
↪ DistancesVectorLidarReference))+1&&numberOfObstacles
↪ <10
                numberOfObstacles=numberOfObstacles+1;
                obstacleLimits(numberOfObstacles,1)=mod(
↪ scanStart-1,length(DistancesVectorLidarReference))+1;
                obstacleLimits(numberOfObstacles,2)=
↪ scanStart;
                % find end of the obstacle and begining
↪ and end of
                % safe space
                while obstacleLimits<=length(
↪ DistancesVectorLidarReference)+firstObstacleEnd+1
                    temp2=mod(obstacleElement-1,length(
↪ DistancesVectorLidarReference))+1;
                    for limitFinder=length(
↪ DistancesVectorLidarReference)+firstObstacleEnd:-1:
↪ obstacleElement
```

```
                        temp=mod(limitFinder-1,length(
↪ DistancesVectorLidarReference))+1;
                        if ((readingsX(temp)-readingsX(
↪ temp2))^2+(readingsY(temp)-readingsY(temp2))^2)<=4*obj
↪ .SafetyRadius^2;
                            obstacleLimits(
↪ numberOfObstacles,2)=temp;
                            break
                        end%if
                    end %for
                    obstacleElement=obstacleElement+1;
                    if obstacleLimits(numberOfObstacles
↪ ,2)==temp2
                        scanStart=obstacleLimits(
↪ numberOfObstacles,2)+1;
                        break % end found
                    end


                end
            end
            [~,index] = ismember(obstacleLimits,index);
```

```
         else
             index=[];
         end
     end %obstacleRecognition
     %% LineTracking
     function [CommandTangentVelocity,
↪ CommandAngularVelocity]=LineTracking(Magnitude,Angle,
↪ obj)
         K1=.7;  %.7
         K2=.1; %.05
         K3=0; %1.5
         MagnitudeRatio=1;
         ErrorX=MagnitudeRatio*Magnitude*cos(Angle-obj.
↪ Theta*pi/180);
         ErrorY=MagnitudeRatio*Magnitude*sin(Angle-obj.
↪ Theta*pi/180);
         ErrorTheta=wrapToPi(Angle-obj.Theta*pi/180);
         FeedbackTangentVelocity=-K1*ErrorX;
         FeedbackAngularVelocity=-K2*sign(obj.
↪ TangentVelocity)*ErrorY-K3*ErrorTheta;
```

```
        CommandTangentVelocity=obj.TangentVelocity*cos(
↪ ErrorTheta)-FeedbackTangentVelocity;
        CommandAngularVelocity=-FeedbackAngularVelocity;
        VelocityRight=CommandTangentVelocity+
↪ CommandAngularVelocity*obj.Width/2;
        VelocityLeft=CommandTangentVelocity-
↪ CommandAngularVelocity*obj.Width/2;
        if abs(VelocityRight)>obj.MaxTangentVelocity
            VelocityRight=obj.MaxTangentVelocity*sign(
↪ VelocityRight);
        end
        if abs(VelocityLeft)>obj.MaxTangentVelocity
            VelocityLeft=obj.MaxTangentVelocity*sign(
↪ VelocityLeft);
        end
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% MoveRobot
    function obj=MoveRobot(obj,Base,TimeStep)
        [Magnitude,Angle]=PotentialFieldController(obj,
↪ Base,TimeStep);
```

```
      [CommandTangentVelocity,CommandAngularVelocity]=
↪ LineTracking(Magnitude,Angle,obj);
      [ForceLeft,ForceRight]=VelocityToForce(
↪ CommandTangentVelocity,CommandAngularVelocity,obj);
      [TangentAcceleration,AngularAcceleration]=
↪ ForceToActuation(ForceLeft,ForceRight,obj);
      obj.TangentVelocity=TangentAcceleration*TimeStep
↪ +obj.TangentVelocity;
      obj.AngularVelocity=AngularAcceleration*TimeStep
↪ +obj.AngularVelocity;
      obj.X=obj.X+obj.TangentVelocity*cos(obj.Theta*pi
↪ /180)*TimeStep;
      obj.Y=obj.Y+obj.TangentVelocity*sin(obj.Theta*pi
↪ /180)*TimeStep;
      obj.Theta=obj.Theta+obj.AngularVelocity*180*
↪ TimeStep/pi;
  end
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%
  function [ForceLeft,ForceRight]=VelocityToForce(
↪ CommandTangentVelocity,CommandAngularVelocity,obj)
```

```
        TangGain=30;
        AngGain=300;
        ForceRight=TangGain*((CommandTangentVelocity-obj
↪ .TangentVelocity)...
            +AngGain*(CommandAngularVelocity-obj.
↪ AngularVelocity)/obj.Width);
        ForceLeft=TangGain*((CommandTangentVelocity-obj.
↪ TangentVelocity)...
            -AngGain*(CommandAngularVelocity-obj.
↪ AngularVelocity)/obj.Width);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%
    function [TangentAcceleration,AngularAcceleration]=
↪ ForceToActuation(ForceLeft,ForceRight,obj)
        Damp=20;
        TangentAcceleration=(ForceRight+ForceLeft-abs(
↪ Damp*obj.TangentVelocity))/obj.Mass;
        AngularAcceleration=((ForceRight-ForceLeft)*obj.
↪ Width-abs(Damp*obj.AngularVelocity))/(obj.Mass*(obj.
↪ Width^2+obj.Length^2)/12);
```

```
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%
        function [XX,YY]=robotReferenceToGlobal(obj,Angles,
    ↪ Distances)
            XX=obj.X+Distances.*cos((Angles+obj.Theta)*pi
    ↪ /180);
            YY=obj.Y+Distances.*sin((Angles+obj.Theta)*pi
    ↪ /180);
        end%function
        %
    ↪ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ↪
        %%
        function [subgoalsDistances,subgoalsTheta]=subgoals(
    ↪ lidarDistances,lidarThetas,obj)
            index=obstacleRecognition(obj,lidarDistances,
    ↪ lidarThetas);
            [robotDistances,robotThetas]=LidarToRobotCenter(
    ↪ lidarDistances,lidarThetas,obj);
            numberOfReadings=length(lidarDistances);
```

```
         numberOfObstacles=size(index,1);
         subgoalsTheta=nan*ones(numberOfObstacles,2);
         subgoalsDistances=nan*ones(numberOfObstacles,2);
         if ~isempty(index)
             if index(1,1)~=0
                 if index(1,2)<index(1,1)
                     index(1,2)=index(1,2)+
↪ numberOfReadings;
                 end
                 tem=mod((index(1,1):index(1,2))-1,
↪ numberOfReadings)+1;
                 temp=real([robotThetas(tem)-asind(obj.
↪ SafetyRadius./robotDistances(tem))]);
                 shift=min(temp);
             else
                 shift=0;
             end
             robotThetas=wrapTo360(robotThetas-shift);
             for i=1:numberOfObstacles
                 if index(i,2)<index(i,1)
```

```
                    index(i,2)=index(i,2)+
↪ numberOfReadings;
                end
                toCompare=index(i,1):1:index(i,2);
                for j=toCompare
                    element=mod(j-1,numberOfReadings)+1;
                    temp1=real(robotThetas(element)-
↪ asind(obj.SafetyRadius/robotDistances(element)));
                    temp2=real(robotThetas(element)+
↪ asind(obj.SafetyRadius/robotDistances(element)));
                    if temp1<-.0001
                        temp1=temp1+360;
                        temp2=temp2+360;
                    end
                    temp3=subgoalsTheta(i,1);
                    temp4=subgoalsTheta(i,2);
                    [subgoalsTheta(i,1),ind]=min([temp3,
↪ temp1,temp2]);
                    if ind~=1
                        subgoalsDistances(i,1)=
↪ robotDistances(element);
```

```
                    end
                    [subgoalsTheta(i,2),ind]=max([temp4,
↪ temp1,temp2]);
                    if ind~=1
                        subgoalsDistances(i,2)=
↪ robotDistances(element);
                    end
                end
            end
            subgoalsTheta=wrapTo360(subgoalsTheta+shift)
↪ ;
        else
            subgoalsTheta=[];
        end
    end%function
    %
↪ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function selectedSubgoal=selectSubgoal(obj,
↪ targetDistance,targetTheta,lidarDistances,lidarThetas)
        for i=1:length(obj.EdgePlot)
            if ishandle(obj.EdgePlot(i))
```

```
                delete(obj.EdgePlot(i));
            end
        end
        [subgoalsDistances,subgoalsTheta]=subgoals(
↪ lidarDistances,lidarThetas,obj);
        targetTheta=wrapTo180(targetTheta);
        for i=1:size(subgoalsTheta,1)
            if subgoalsTheta(i,1)>subgoalsTheta(i,2)
                subgoalsTheta(i,1)=subgoalsTheta(i,1)
↪ -360;
            end
        end
        subgoalsTheta2=wrapTo180(subgoalsTheta);
        difference=abs(targetTheta-subgoalsTheta2)+
↪ subgoalsDistances.*abs(wrapTo180(subgoalsTheta2))*.01;
        selectedSubgoal=[targetDistance,targetTheta];
        if ~isempty(subgoalsDistances)
            for i=1:size(subgoalsDistances,1)
                diff=wrapTo180(subgoalsTheta(i,:)-
↪ selectedSubgoal(2));
                if diff(1)>diff(2)
```

```
                    diff(1)=diff(1)-360;
                end
                if (sign(diff(1)*diff(2))==-1)&&
↪ selectedSubgoal(1)>sum(subgoalsDistances(i,:))/2
                    [~,Indx]=min([difference(i,1),
↪ difference(i,2)]);
                    selectedSubgoal=[subgoalsDistances(i
↪ ,Indx),subgoalsTheta2(i,Indx)];
                elseif (subgoalsTheta2(i,2)<
↪ subgoalsTheta2(i,1)||subgoalsTheta(i,2)>360)&&
↪ targetDistance>sum(lidarDistances(i,:))/2
                    if selectedSubgoal(2)<subgoalsTheta2
↪ (i,2)&&selectedSubgoal(2)>subgoalsTheta2(i,1)
                        [~,Indx]=max([difference(i,1),
↪ difference(i,2)]);
                        selectedSubgoal=[
↪ subgoalsDistances(i,Indx),subgoalsTheta2(i,Indx)];
                    end
                end%if

            end%for
```

```
        end

        for i=1:size(subgoalsDistances,1)
            start=subgoalsTheta2(i,2);
            stop=subgoalsTheta2(i,1);
            angels=[start,stop];
            start=subgoalsDistances(i,2);
            stop=subgoalsDistances(i,1);
            distances=[start,stop];
            [XX,YY]=robotReferenceToGlobal(obj,angels,
↪ distances);
            obj.EdgePlot(i)=plot(XX',YY','c:o','
↪ MarkerSize',10,'MarkerFaceColor','c','LineWidth',1);
        end%for


        [subgoalX,subgoalY]=robotReferenceToGlobal(obj,
↪ selectedSubgoal(2),selectedSubgoal(1));
        obj.EdgePlot(size(subgoalsDistances,1)+1)=
↪ rectangle('Position',[subgoalX-5,subgoalY-5,10,10],...
↪     %shows subgoal
```

```
            'Curvature',[1,1],'LineWidth',3,'edgecolor',
↪ 'm');

    end%function
    %
↪ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%
    function Direction=GapFinding(LidarReadingAngles,
↪ LidarReadingDistances,obj)

        DistanceFromTarget=sqrt((obj.TargetX-obj.X)^2+(
↪ obj.TargetY-obj.Y)^2);
        TargetDirection=wrapTo180(atan2((obj.TargetY-obj
↪ .Y),(obj.TargetX-obj.X))*180/pi-obj.Theta);
        NumberOfReadings=length(LidarReadingAngles);  %
↪ from LIDAR
        SeeObstacle=zeros(NumberOfReadings+2,1);
        for i=1:NumberOfReadings  % does it see obstacle
            if (LidarReadingDistances(i)<obj.
↪ DistanceRange)&&(LidarReadingDistances(i)<
↪ DistanceFromTarget)
```

```
                 SeeObstacle(i+1)=1;
             else SeeObstacle(i+1)=0;
             end%if
         end%for
         targetTheta=TargetDirection;
         targetDistance=DistanceFromTarget;

         selectedSubgoal=selectSubgoal(obj,targetDistance
↪  ,targetTheta,LidarReadingDistances,LidarReadingAngles)
↪  ;
         if targetDistance==selectedSubgoal(1)&&
↪  targetTheta==selectedSubgoal(2)

         else
             targetDistance=real(selectedSubgoal(1));
             targetTheta=real(selectedSubgoal(2));
         end%if

   Direction=wrapTo180(targetTheta+obj.Theta)*pi/180;
     end     %GapFinding
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    end %end methods

end
```

B.4 LIDAR Object Definition

```
classdef Lidar < handle

    properties
        Type;
        DistanceRange=250;
        AngularRange=260;
        ScanStepSize=1;
        LidarPlot;
    end

    methods

        function obj=Lidar()
    end

    function [LidarReadingAngles,LidarReadingDistances]=
↪ LidarData(Robot,Base)
        LidarPositionX=Robot.X+cos(Robot.Theta*pi/180)*
↪ Robot.LidarOffCenter;
        LidarPositionY=Robot.Y+sin(Robot.Theta*pi/180)*
↪ Robot.LidarOffCenter;
        Map=Base.Matrix;
        MapSize=size(Map);
        LineStep=1;

        if ishandle(Robot.LidarPlot)
            delete(Robot.LidarPlot)
        end
        LidarReadingAngles=(-Robot.AngularRange/2+Robot.
↪ ScanStepSize/2:Robot.ScanStepSize:Robot.AngularRange
↪ /2-Robot.ScanStepSize/2)';
        LidarReadingDistances=zeros(Robot.AngularRange/
↪ Robot.ScanStepSize,1);
```

```
         for i=1:Robot.AngularRange/Robot.ScanStepSize
             m=(LidarReadingAngles(i)+Robot.Theta)*pi
↪ /180;
             TravelStep=1;
             exceed=0;

             while exceed==0
                 LightTravel=TravelStep*LineStep;
                 LightedPix(TravelStep,:)=[int16(
↪ LidarPositionX+LightTravel*cos(m)),int16(
↪ LidarPositionY+LightTravel*sin(m))];

                 if LightedPix(TravelStep,1)>0&&
↪ LightedPix(TravelStep,1)<MapSize(2)&&LightedPix(
↪ TravelStep,2)>0&&LightedPix(TravelStep,2)<MapSize(1) %
↪ if inside map

                     if Map(LightedPix(TravelStep,2),
↪ LightedPix(TravelStep,1))==1  %if hit a wall
                         LidarReadingDistances(i)=
↪ LightTravel;
```

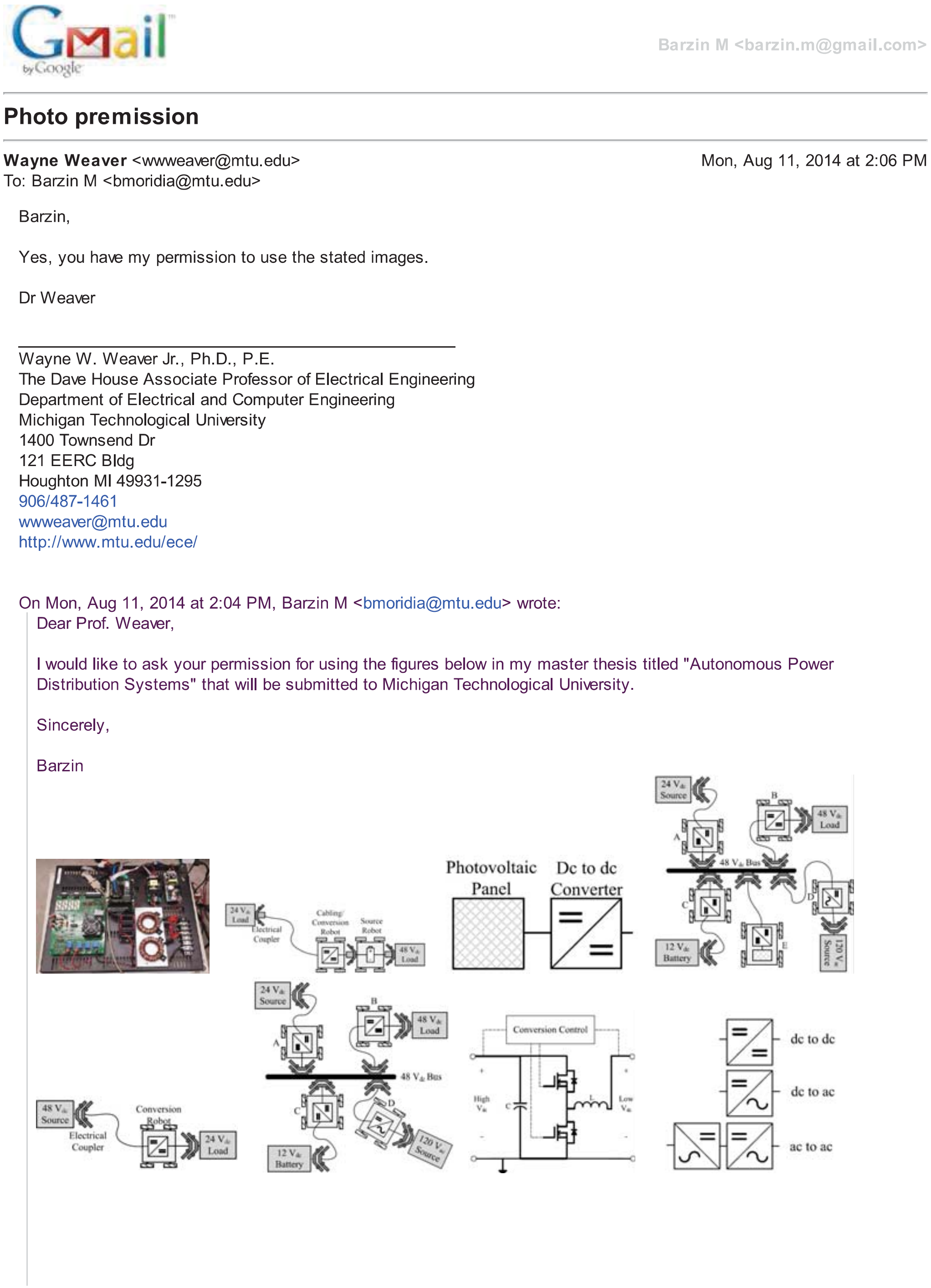
```
                        exceed=1;
                    end
                    TravelStep=TravelStep+1;

                    if LightTravel>Robot.DistanceRange
                        LidarReadingDistances(i)=
↪  LightTravel-LineStep;
                        exceed=1;
                    end

                else
                    LidarReadingDistances(i)=LightTravel
↪  ;
                    exceed=1;
                end
            end
```

```
                Robot.LidarPlot(i)=line([LidarPositionX,
   ↪ LidarPositionX+LidarReadingDistances(i)*cos((
   ↪ LidarReadingAngles(i)+Robot.Theta)*pi/180)],[
   ↪ LidarPositionY,LidarPositionY+LidarReadingDistances(i)
   ↪ *sin((LidarReadingAngles(i)+Robot.Theta)*pi/180)],'
   ↪ Color','y','LineWidth',.1,'LineStyle',':');
            end
        end

        function PlotLidar(Sensor)
        end

    end

end
```

Appendix C

Letters of Permission

Permission letter for figures 4.6, 4.7,4.8,4.9,4.10, 4.11, 4.1, refFirstStage, 4.3, and 4.4.



Barzin M <barzin.m@gmail.com>

Photo premission

Wayne Weaver <wwweaver@mtu.edu> Mon, Aug 11, 2014 at 2:06 PM
To: Barzin M <bmoridia@mtu.edu>

Barzin,

Yes, you have my permission to use the stated images.

Dr Weaver

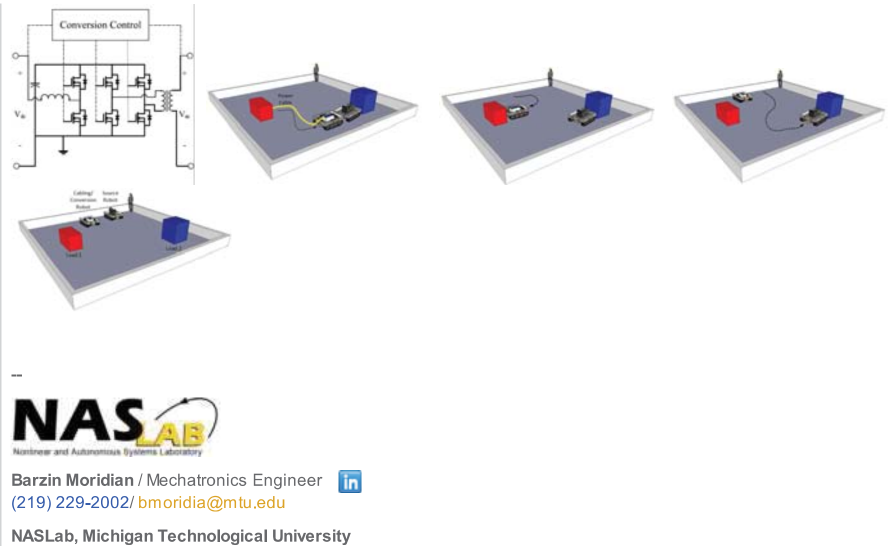
Wayne W. Weaver Jr., Ph.D., P.E.
The Dave House Associate Professor of Electrical Engineering
Department of Electrical and Computer Engineering
Michigan Technological University
1400 Townsend Dr
121 EERC Bldg
Houghton MI 49931-1295
906/487-1461
wwweaver@mtu.edu
http://www.mtu.edu/ece/

On Mon, Aug 11, 2014 at 2:04 PM, Barzin M <bmoridia@mtu.edu> wrote:

> Dear Prof. Weaver,
>
> I would like to ask your permission for using the figures below in my master thesis titled "Autonomous Power Distribution Systems" that will be submitted to Michigan Technological University.
>
> Sincerely,
>
> Barzin

24 V_{dc} Load
Electrical Coupler
Cabling/ Conversion Robot
Source Robot
48 V_{dc} Load
Photovoltaic Panel
Dc to dc Converter
24 V_{dc} Source
A
B
C
D
E
48 V_{dc} Load
48 V_{dc} Bus
12 V_{dc} Battery
120 V_{ac} Source
48 V_{dc} Source
Conversion Robot
24 V_{dc} Load
Conversion Control
High V_{dc}
Low V_{dc}
C
L
dc to dc
dc to ac
ac to ac

Figure C.1: Figure permission from Wayne W. Weaver



--

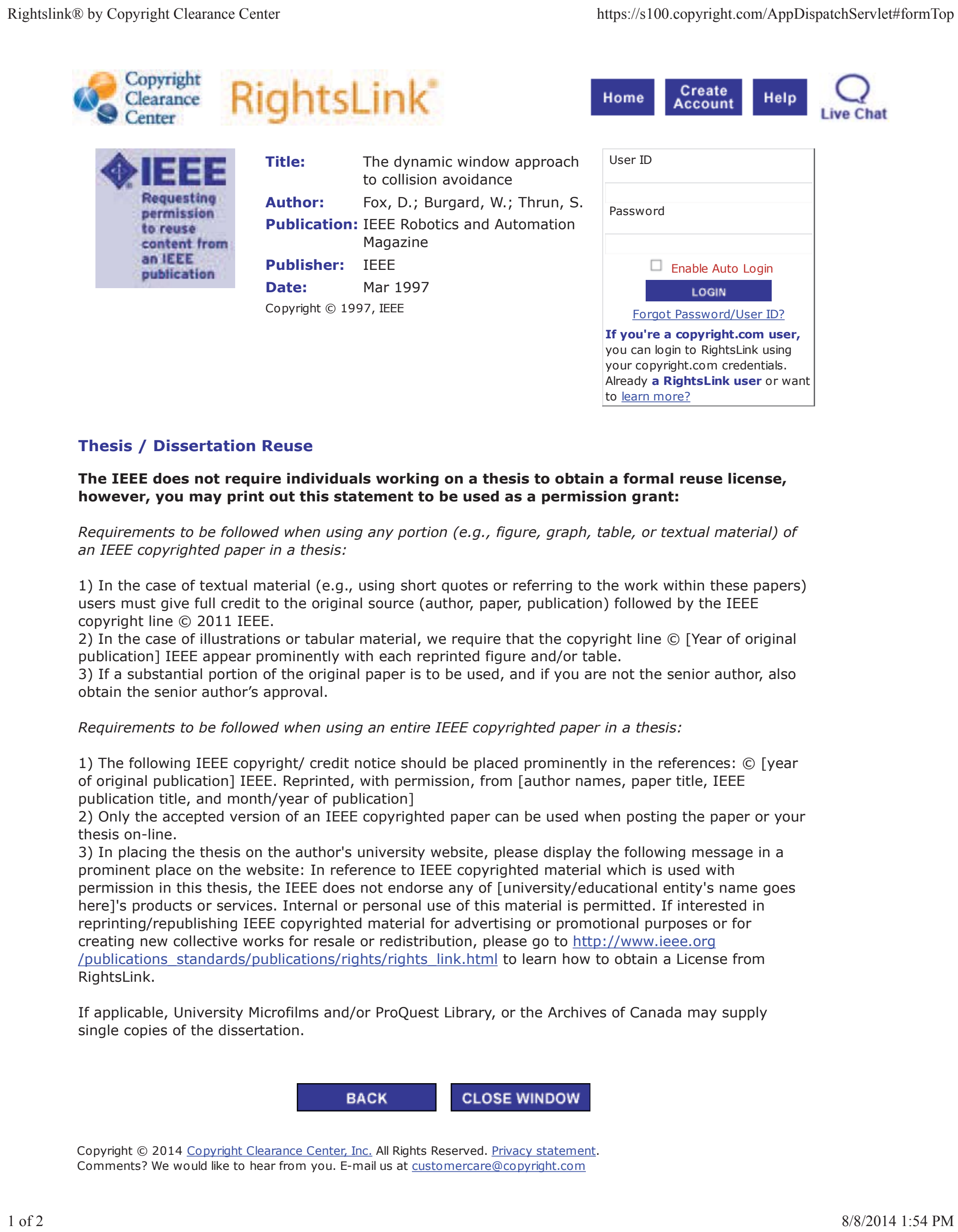
NASLAB
Nonlinear and Autonomous Systems Laboratory

Barzin Moridian / Mechatronics Engineer
(219) 229-2002/ bmoridia@mtu.edu

NASLab, Michigan Technological University

Figure C.2: Figure permission from Wayne W. Weaver; Continued

Permission letter for figures 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, and2.11.



Rightslink® by Copyright Clearance Center https://s100.copyright.com/AppDispatchServlet#formTop

Title: The dynamic window approach to collision avoidance
Author: Fox, D.; Burgard, W.; Thrun, S.
Publication: IEEE Robotics and Automation Magazine
Publisher: IEEE
Date: Mar 1997
Copyright © 1997, IEEE

User ID

Password

☐ Enable Auto Login

Forgot Password/User ID?

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials. Already **a RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Copyright © 2014 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement.
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

1 of 2 8/8/2014 1:54 PM

Figure C.3: IEEE figures permission