



**NAVAL
POSTGRADUATE
SCHOOL**

MONTEREY, CALIFORNIA

THESIS

**ASSESSING THE EFFECTIVENESS OF A COMBAT
UGV SWARM IN URBAN OPERATIONS**

by

Boon Hong Aaron Teow

September 2018

Thesis Advisor:

Oleg A. Yakimenko

Second Reader:

Fotis A. Papoulias

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2018	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE ASSESSING THE EFFECTIVENESS OF A COMBAT UGV SWARM IN URBAN OPERATIONS			5. FUNDING NUMBERS	
6. AUTHOR(S) Boon Hong Aaron Teow				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Due to its complexity, an urban area is a challenging multi-dimensional environment for ground warfare. Recent technological advancements have enabled militaries to utilize different-size unmanned ground vehicles (UGV) to support a variety of missions. This thesis presents guidance algorithms for a search and kill mission developed for some generic UGV swarms, which may be an attractive application, particularly for smaller UGVs operating in an urban environment. Through a series of computer simulations, the research evaluates the feasibility and effectiveness of the algorithms in executing such a mission in indoor and outdoor urban environments. The developed simulation allows varying many parameters, thus achieving closeness to the real-world situation when different environments, platforms, sensors, and weapons are used. Computer simulations presented in this paper may also assist military leaders in choosing key mission parameters to maximize the outcome of potential future engagements.				
14. SUBJECT TERMS swarm, unmanned ground vehicle, UGV, Particle Swarm Optimization, modeling and simulation			15. NUMBER OF PAGES 187	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**ASSESSING THE EFFECTIVENESS OF A COMBAT UGV SWARM IN URBAN
OPERATIONS**

Boon Hong Aaron Teow
Major, Singapore Army
B.Eng., Nanyang Technological University, 2010

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
September 2018**

Approved by: Oleg A. Yakimenko
Advisor

Fotis A. Papoulias
Second Reader

Ronald E. Giachetti
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Due to its complexity, an urban area is a challenging multi-dimensional environment for ground warfare. Recent technological advancements have enabled militaries to utilize different-size unmanned ground vehicles (UGV) to support a variety of missions. This thesis presents guidance algorithms for a search and kill mission developed for some generic UGV swarms, which may be an attractive application, particularly for smaller UGVs operating in an urban environment. Through a series of computer simulations, the research evaluates the feasibility and effectiveness of the algorithms in executing such a mission in indoor and outdoor urban environments. The developed simulation allows varying many parameters, thus achieving closeness to the real-world situation when different environments, platforms, sensors, and weapons are used. Computer simulations presented in this paper may also assist military leaders in choosing key mission parameters to maximize the outcome of potential future engagements.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	BACKGROUND	1
B.	OBJECTIVES	4
C.	RESEARCH QUESTIONS	4
D.	SCOPE	5
1.	Functional Analysis and Allocation.....	5
2.	Functional Flow Block Diagram.....	6
E.	ASSUMPTIONS.....	7
II.	LITERATURE REVIEW	9
A.	SEARCH ALGORITHMS.....	9
1.	Exhaustive Search.....	9
2.	Heuristic Search	13
3.	Summary of Search and Swarm Optimization Algorithms Used in this Research.....	15
B.	APPLICATIONS OF SWARM ALGORITHMS	16
C.	MACHINE VISION.....	17
D.	HOLONOMIC BEHAVIOR.....	18
III.	MODELING	19
A.	MOTION PRIMITIVES	19
B.	MOTION CONSTRAINTS	21
1.	Least Visited Cell Guidance.....	21
2.	Advanced Least Visited Cell Guidance.....	23
C.	PARTICLE SWARM OPTIMIZATION	25
D.	ENGAGEMENT RULES.....	26
E.	OPERATIONAL ENVIRONMENT	27
IV.	SEARCH PHASE STUDY.....	29
A.	EFFECT OF SWARM SIZE	29
B.	AREA COVERAGE VERSUS THE NUMBER OF ITERATIONS	31
C.	SWARM SIZE VERSUS NUMBER OF ITERATIONS	34
D.	EFFECT OF STARTING CONFIGURATION	36
1.	Effect of Swarm Size on Starting Configurations.....	38
2.	Effect of Maximum Number of Iterations on Starting Configurations.....	39

E.	EFFECT OF THE COLLISION AVOIDANCE CONSTRAINT	40
F.	EFFECT OF THE NON-HOLONOMICITY CONSTRAINT	43
G.	URBAN OUTDOOR SEARCH OPERATIONS	51
1.	Effect of Various Starting Configurations.....	52
2.	Effect of the Non-Holonomicity Constraint.....	53
H.	INDOOR SEARCH OPERATIONS	56
I.	EFFECTIVENESS OF ALVC GUIDANCE	59
1.	Holonomic Drive	60
2.	Environment.....	62
V.	STUDY OF THE TRACK AND ENGAGE PHASE	63
A.	EFFECTIVENESS OF ADDED PSO GUIDANCE	64
B.	EFFECTS OF VARYING DETECTION RANGE	68
C.	EFFECTS OF THE HOLONOMICITY CONSTRAINT DURING TRACKING	69
D.	EFFECTS OF PROBABILITY OF KILL	72
E.	EFFECTS OF KILL DISTANCE	78
F.	EFFECTS OF KILL SEQUENCE.....	79
G.	URBAN OUTDOOR ENGAGEMENTS	82
H.	INDOOR ENGAGEMENTS.....	84
I.	EFFECTIVENESS OF ALVC GUIDANCE	88
J.	LIMITATIONS OF ALVC GUIDANCE	90
VI.	CONCLUSION	93
A.	SUMMARY	93
B.	MAIN FINDINGS	93
C.	RECOMMENDATIONS FOR FUTURE WORK.....	95
APPENDIX A. SEARCH PHASE WITH LVC GUIDANCE		97
APPENDIX B. SEARCH PHASE WITH ALVC GUIDANCE.....		105
APPENDIX C. TRACK AND ENGAGE PHASE WITH LVC GUIDANCE.....		117
APPENDIX D. TRACK AND ENGAGE PHASE WITH ALVC GUIDANCE.....		133
APPENDIX E. COORDINATES FOR OUTDOOR OBSTACLES		151
APPENDIX F. COORDINATES FOR INDOOR OBSTACLES.....		153

LIST OF REFERENCES	155
INITIAL DISTRIBUTION LIST	159

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	DoD Army UGV campaign plan. Source: U.S. Department of Defense (2011).....	2
Figure 2.	DoD Army UGV capability timeline. Source: U.S. Department of Defense (2011).....	2
Figure 3.	Functional decomposition of swarm combat UGV system.	5
Figure 4.	FFBD for swarm combat UGVs.	7
Figure 5.	Robot moving in a “lawn mower” pattern through the cells. Source: Galceran and Carreras (2013).	10
Figure 6.	Broken up cells in the trapezoidal decomposition. Source: Galceran and Carreras (2013).....	11
Figure 7.	Assigned values for each cell using the wavefront algorithm. Source: Zelinsky et al. (1993).	12
Figure 8.	Path of complete coverage using wavefront algorithm. Source: Zelinsky et al. (1993).	12
Figure 9.	Agent surrounding cells.	20
Figure 10.	Agent’s surrounding visited and unvisited cells.	21
Figure 11.	Surroundings visited, unvisited, and occupied cells.	22
Figure 12.	Surrounding cells with a non-holonomicity constraint of 90 degrees.	23
Figure 13.	Surrounding cells with a non-holonomicity constraint of 90 degrees and obstacles.	23
Figure 14.	Illustrations of improved algorithm with non-holonomicity constraint of 90 degrees.	24
Figure 15.	Example of a swarm pursuing two Red agents.	26
Figure 16.	Model simulation of Impossible City at Fort Ord, California (right) and Google map view (left).	28
Figure 17.	Indoor floorplan of a room.....	28
Figure 18.	Effect of swarm size on area coverage.	29

Figure 19.	Average number of visits and number of cells for a swarm size of 10 agents with 1,000 iterations.	30
Figure 20.	Average number of visits and number of cells for a swarm size of 70 agents with 1,000 iterations.	31
Figure 21.	Effects of the maximum number of iterations on area coverage.	32
Figure 22.	Trajectory plot of 19 percent coverage for 20 agents on 200 iterations.....	33
Figure 23.	Trajectory plot of 70 percent coverage for 20 agents on 1,000 iterations.....	33
Figure 24.	Effects of swarm size on area coverage with iteration comparison.....	34
Figure 25.	Estimated swarm size or number of iterations needed to achieve required area coverage.	35
Figure 26.	Various starting configurations.....	36
Figure 27.	Snapshot of Corner starting configuration.....	36
Figure 28.	Snapshot of Center starting configuration.	37
Figure 29.	Snapshot of Two-Corners starting configuration.....	37
Figure 30.	Snapshot of Four-Corners starting configuration.	38
Figure 31.	Snapshot of Row starting configuration.	38
Figure 32.	Coverage of various starting configurations by swarm size.	39
Figure 33.	Coverage of various starting configurations by number of iterations (duration).....	40
Figure 34.	Maximum and minimum distances between any two agents.....	41
Figure 35.	Effect of starting positions on coverage when incorporating collision avoidance.	42
Figure 36.	Heat map comparison of 50 iterations for collision avoidance (left) and without collision avoidance (right).	43
Figure 37.	Non-holonomicity constraint of 90 degrees.....	44
Figure 38.	Non-holonomicity constraint of 180 degrees.....	44

Figure 39.	Non-holonomicity constraint of 270 degrees.....	44
Figure 40.	Non-holonomicity constraint of 90 degrees.....	46
Figure 41.	Non-holonomicity constraint of 180 degrees.....	47
Figure 42.	Non-holonomicity constraint of 270 degrees.....	48
Figure 43.	No non-holonomicity angle constraint.....	49
Figure 44.	Effect on various non-holonomic angle constraints.....	50
Figure 45.	Effect of a non-holonomicity constraint of 180 degrees and various starting configurations.	51
Figure 46.	Starting configurations of UGV agents for urban outdoor operations.....	52
Figure 47.	Effect of starting configuration on area coverage for urban operation.	53
Figure 48.	Effect of non-holonomic angle restriction on area coverage for urban scenario.	54
Figure 49.	Holonomic (left) versus non-holonomic drive with a 90-degree constraint (right).....	55
Figure 50.	Starting position (entrance) to the indoor environment.	57
Figure 51.	Effect of non-holonomic drive on area coverage for indoor operations.....	58
Figure 52.	Snapshot of the last 100 iterations of a simulated run using ALVC guidance.	60
Figure 53.	Infinite circle loop around an unvisited square.....	61
Figure 54.	Effect of non-holonomic drive on the ALVC algorithm.	61
Figure 55.	Comparison between results of LVC and ALVC algorithms for three environments.....	62
Figure 56.	Starting configurations of Blue and Red forces for open space (left), outdoor (center), and indoor (right) urban operations.	63
Figure 57.	Trajectory comparison between LVC (left) and PSO (right) guidance during the track and engage phase.....	65
Figure 58.	Number of iterations needed for a battle with and without PSO guidance.	66

Figure 59.	Number of casualties with and without PSO guidance.....	66
Figure 60.	ANOVA table for the number of iterations needed for a battle with and without PSO guidance.....	67
Figure 61.	Testing for significant different result.	67
Figure 62.	Number of iterations corresponding to various detection ranges.	68
Figure 63.	Number of casualties corresponding to various detection ranges.....	69
Figure 64.	Comparison of holonomic and non-holonomic drive during tracking phase under PSO guidance.....	70
Figure 65.	Snapshot of two consecutive iterations during tracking phase with a 90-degree non-holonomicity constraint.	71
Figure 66.	Snapshot of two consecutive iterations during tracking phase with holonomic drive.	71
Figure 67.	Effects of time with fixed enemy offensive capability of 0.1 and varying UGV agents' offensive capability.	72
Figure 68.	Effects of time with fixed enemy offensive capability of 0.5 and varying UGV agents' offensive capability.	73
Figure 69.	Effects of time with fixed enemy offensive capability of 0.9 and varying UGV agents' offensive capability.	73
Figure 70.	Effects on casualty rate with fixed enemy offensive capability of 0.1 and varying UGV agents' offensive capability.....	74
Figure 71.	Effects on casualty rate with fixed enemy offensive capability of 0.5 and varying UGV agents' offensive capability.....	75
Figure 72.	Effects on casualty rate with fixed enemy offensive capability of 0.9 and varying UGV agents' offensive capability.....	75
Figure 73.	Effects of kill distance on number of iterations.	78
Figure 74.	Effects of kill distance on number of casualties.	79
Figure 75.	Effects of kill sequence on the number of iterations.....	80
Figure 76.	Effects of kill sequence on the number of Blue force casualties.	81
Figure 77.	Effects of kill sequence on the number of Red force casualties.	81

Figure 78.	Starting configuration for outdoor urban operation.	82
Figure 79.	Effects on number of iterations with and without PSO guidance for outdoor operation.	83
Figure 80.	Effects on number of casualties with and without PSO guidance for outdoor operation.	84
Figure 81.	Starting configurations for indoor operation.....	85
Figure 82.	Effects on number of iterations, with and without PSO guidance, for indoor operation.	86
Figure 83.	Effects on number of casualties, with and without PSO guidance, for indoor operation.	86
Figure 84.	Inability to avoid obstacles with low detection range (left) and high detection range (right).	87
Figure 85.	Effects of LVC with ALVC algorithms, with and without PSO guidance, on number of iterations.....	89
Figure 86.	Effects of LVC with ALVC algorithms, with and without PSO guidance, on number of casualties.	90
Figure 87.	Limitations of the improved LVC algorithm in urban and indoor operations.	91

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Technology areas that require growth to meet future capabilities of U.S. Army UGV campaign plan. Source: U.S. Department of Defense (2011).....	3
Table 2.	Summary of search algorithms	15
Table 3.	Comparison of the improvement (area coverage) in open space and outdoor urban environments with the effect of non-holonomicity constraint.....	56
Table 4.	Comparison of the improvement (area coverage) in open space and indoor environments with effects of non-holonomic constraint.	58
Table 5.	Simulation results for $PkR \rightarrow B = 0.5$ and $PkB \rightarrow R = 0.9$	76
Table 6.	Summary of results for $PkR \rightarrow B = 0.5$ and $PkB \rightarrow R = 0.9$	77
Table 7.	Summary of algorithms and input parameters investigated.....	93

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ACO	Ant Colony Optimization
ALVC	Advance Least Visited Cell
ANOVA	Analysis of Variance
DoD	Department of Defense
FFBD	Functional Flow Block Diagram
LSD	Least Significant Difference
LVC	Least Visited Cell
PSO	Particle Swarm Optimization
RS JPO	Robotic Systems Joint Project Office
MM-UGV	Multi Mission Unmanned Ground Vehicle
UGV	Unmanned Ground Vehicle

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Fighting in urban areas is extremely complex and challenging due to the multi-story structures, new engagement conditions, as well as the consideration of civilian-military relations. Recent technological advancements have enabled the military to employ robotic platforms such as explosive ordinance disposal, heavy items loading, repairing ground conditions under fire, to help overcome operational challenges in the urban environment (Gage 1995). An emerging trend in the realm of military robotics is swarm robotics. Based on the unmanned ground systems roadmap report by the Robotic Systems Joint Project Office (RS JPO) of the U.S. Department of Defense (DoD) published in 2011, there are plans to develop armed UGVs with combat abilities within the next 25 years (Department of Defense 2011, 41).

This thesis assesses the suitability of three algorithms (Table 1), the Least Visited Cell (LVC) guidance, the Advanced Least Visited Cell (ALVC) guidance, and the Particle Swarm Optimization (PSO) algorithm, in three different environments—open space, outdoor, and indoor—in meeting a UGV’s mission of search and destroy. The mission of the UGV is broken down into two phases. The first phase is the search phase and its measure of effectiveness is area coverage. The second phase is the track and engage phase and its measures of effectiveness are the time (the number of iterations) required to end an engagement as well as the number of casualties for Blue and Red forces.

Table 1. Summary of input parameters investigated in this thesis.

Phase	Algorithm	Input Parameters
Search	LVC	Number of UGV agents
	LVC	Number of maximum iterations
	LVC	Starting configuration
	LVC	Collision avoidance constraints
	LVC, ALVC	Non-holonomicity constraints
	LVC, ALVC	Outdoor and indoor urban environments
Track and Engage	LVC, PSO	Non-holonomicity constraints
	LVC, PSO	Detection range
	LVC, PSO	Probability of kill
	LVC, PSO	Kill distance
	LVC, PSO	Kill sequence
	LVC, ALVC, PSO	Outdoor and indoor urban environments

The LVC guidance algorithm that is develop in this thesis works well for all three operations; open space, outdoor, and indoor urban operation. The introduction of the PSO algorithm further enhances and reduces the time taken to locate targets during the track and engage phase by approximately five times. Nonetheless, the PSO algorithm encounters difficulty in indoor operations where it is unable to overcome obstacles between the UGV agents and the detected enemy agent. As the PSO algorithm does not change the engagement sequence or probability of kill, it does not affect the number of casualties sustained. The ALVC guidance algorithm developed as an improvement to the LVC guidance works well and has a significant impact on area coverage, but only in the search phase. Similar to the PSO algorithm, the ALVC guidance algorithm's inability to overcome obstacles makes it unsuitable for outdoor and indoor urban operations. Thus, further modifications for the PSO and ALVC guidance algorithm is required.

Analysis of the simulation results reveals that increasing the number of UGVs would assist in locating targets in a shorter period of time and would also lead to a higher probability of win in the track and engage phase. Analysis also shows that the availability of multiple entry points into the operational area is beneficial as it allows the UGVs to locate the enemy in less time. Further, results from the introduction of the non-holonomicity constraint show that non-holonomic drive improves area coverage and thus allows the UGVs to locate targets in a shorter amount of time. The non-holonomicity constraint, however, proves to be a disadvantage for UGVs tracking a moving target. It is more beneficial to increase detection range, as better situational awareness for the UGVs allows for earlier activation of the PSO algorithm, which would reduce the total engagement time.

The three input parameters affecting the number of casualties are the probability of kill, kill distance, and kill sequence. The number of casualties of Blue forces increases or decreases depending on these three factors. Blue forces suffer fewer casualties with a higher probability of kill of Red forces, Blue forces require a longer kill distance, and Blue forces are first to engage in the battle.

In this thesis, modeling and simulations are done in MATLAB. The kinematics for all agents are defined in Equations (1) and (2), where i represents iterations and Δx and Δy represent the change in the respective coordinates.

$$\mathbf{P}_{i,j+1} = \mathbf{P}_{ij} + \Delta \mathbf{P}_{ij} \quad (1)$$

The concept of modeling follows a grid-based system in which the operational area is broken down into 100 by 100 cells. In each iteration, each agent would determine its next position by evaluating the immediate eight surrounding cells. Depending on the input parameters, such as non-holonomicity drive behaviors and collision avoidance, and environment conditions, such as obstacles, some of the surrounding cells would be restricted. Depending on the algorithm choice, some of the unrestricted cells would be preferred over the others. Engagement between the UGV agents and the enemy agents was modeled as probability events.

References

- Gage, Douglas W. 1995. "UGV History 101: A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts." *Unmanned Systems Magazine* 13, no. 3 (January): 9–32. <http://www.dtic.mil/dtic/tr/fulltext/u2/a422845.pdf>.
- United Nations, Department of Economic and Social Affairs, Population Division. 2014. *World Urbanization Prospects: The 2014 Revision, Highlights (ST/ESA/SER.A/352)*. <https://esa.un.org/unpd/wup/publications/files/wup2014-highlights.pdf>.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to express my deepest gratitude to Professor Oleg Yakimenko for his guidance. His support and commitment allowed me to develop my research in the area of my interest and complete this challenging but rewarding journey at the Naval Postgraduate School. I am extremely proud of the work we managed to accomplish.

I am truly grateful to the Singapore Armed Forces for providing me this opportunity to further my studies and to enrich my life experience. I have been blessed to work alongside the 2017 cohort of the Temasek Defense Systems Institute.

Finally, I would like to thank Claire, for her unwavering love and support throughout this journey.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The global trend of urbanization that began after World War II continues to grow rapidly (Glenn 1996, 2). In 2014, 54 percent of the world's population resided in urban areas, compared to only 30 percent in 1950, and the United Nations estimates that by 2050, that number will reach 66 percent (United Nations 2014). This global trend necessarily contributes to a shift in the characteristics of any future potential conflicts, and as a result, in the way urban warfare would be conducted.

Fighting in urban areas is extremely complex and challenging. The third dimension in urban areas, such as subterranean and multi-story structures, affects the line of sight and engagement conditions, thus increasing the complexity of the environment. In addition, the presence of civilians introduces constraints, such as reduced air or artillery support for ground troops, to minimize non-combatant casualties and collateral damage. The complexity of the environment requires better situational awareness, equipment, and training to overcome these challenges.

Technological advancements in recent years have equipped armed forces to meet these challenging demands. Such advancements include military robotic platforms, which are now frequently employed by troops for explosive ordinance disposal, loading and carrying heavy items, and repairing ground conditions under fire (Gage 1995). Furthermore, according to the 2011 unmanned ground systems roadmap report by the Robotic Systems Joint Project Office (RS JPO) of the United States Department of Defense (DoD), there are plans to develop armed unmanned ground vehicles (UGV) with combat abilities within the next 25 years (Department of Defense 2011, 41). Figure 1 shows a variety of the UGVs being developed.

One of the emerging concepts that the RS JPO is actively tracking is the Multi-Mission Unmanned Ground Vehicle (MM-UGV). MM-UGVs possess armed unmanned capability as well as the capability to deal with improvised explosive devices. As seen in

Figure 2, full autonomy for an unmanned combat ground vehicle such as the MM-UGV is a far-term capability anticipated by the U.S. Army.

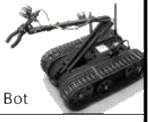















Soldier Transportable	Vehicle Transportable	Self Transportable	Appliqué
 <p>Crew Served Bot</p>	 <p>Mounted or Towed</p> <p>Man Transportable Robot System (MTRS) POR</p>	 <p>Soldier Follower – IBCT</p> <p>Squad Mission Equipment Transport (S-MET) CDD</p>	 <p>Remote Operation</p> <p>Husky Mounted Detection System (HMDS) POR</p>
 <p>Small Bot</p> <p>Small Unmanned Ground Vehicle (SUGV) CDD</p>		 <p>Medium Wingman – SBCT</p> <p>Multi-Mission Unmanned Ground Vehicle (MM-UGV) CDD</p>	 <p>Supervised Autonomy</p> <p>Convoy Active Safety Technology (CAST) CDD</p>
 <p>Micro Bot</p>	 <p>Armed</p> <p>Battlefield Extraction Assist Robot (BEAR) Initiative</p>	 <p>Heavy Wingman – HBCT</p>	 <p>Full Autonomy</p> <p>Combat Autonomous Mobility System (CAMS) JCTD</p>
 <p>Nano Bot</p>	 <p>Humanoid</p> <p>Battlefield Extraction Assist Robot (BEAR) Initiative</p>	 <p>Squad Member</p>	 <p>Exoskeleton</p> <p>Exoskeleton (XOS) CDD</p>

Figure 1. DoD Army UGV campaign plan. Source: U.S. Department of Defense (2011).

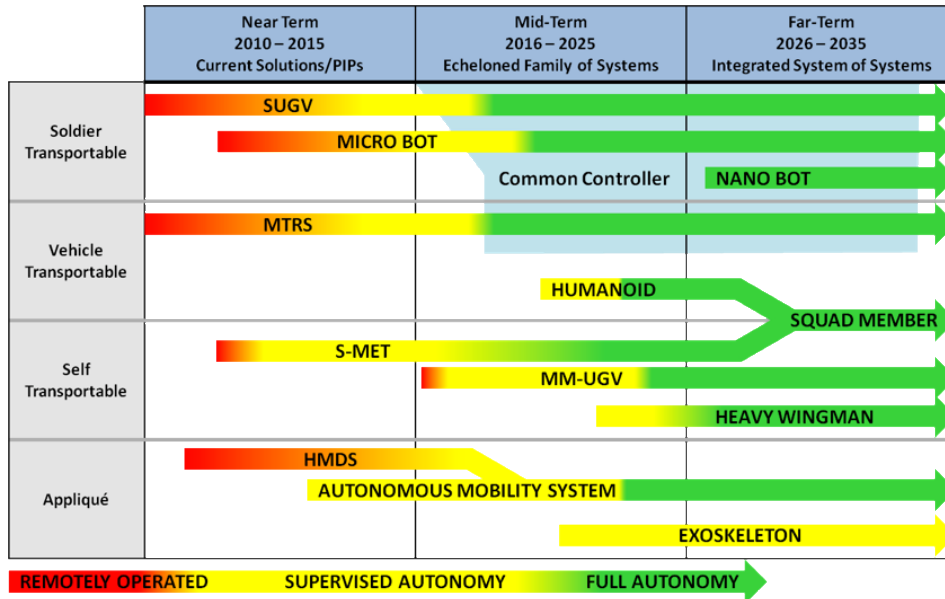


Figure 2. DoD Army UGV capability timeline. Source: U.S. Department of Defense (2011).

To meet the future capabilities requirement as identified in the U.S. Army UGV campaign plan (Figure 1), unmanned ground vehicles require further technological advancement. Table 1, which is drawn from the RS JPO report, summarizes the technology growth needed.

Table 1. Technology areas that require growth to meet future capabilities of U.S. Army UGV campaign plan. Source: U.S. Department of Defense (2011).

Priority	Technology Area
1	Autonomy
1	Obstacle detection and Avoidance
2	Interoperability
2	Commonality
3	Increased NLOS and LOS capability (COMS)
3	Improved Culvert Interrogation Ability
4	Frequency Spectrum Adaptability
5	Extended Mission Duration
6	COMSEC Encryption Capability
6	Net-Ready KPP
7	Common Controller
8	Improved Optics
9	Health Management System
10	Render Useless Mechanism
11	Layered, Escalating Defense Mechanisms
12	Audio Directional Detection
13	Explosive Detection
14	Embedded Training Capability
15	Location Reporting
16	Integrated Tool Kit
17	Dismounted Mission Enabling Robotics

For the micro- and nano-bots depicted in Figure 1, swarming is considered to be one of the most promising capabilities to be developed, according to Vasily Kashin of the Higher School of Economics in Moscow and an expert on China's military (Feng and

Clover 2017). Swarm intelligence is an artificial intelligence discipline that consists of a multi-agent system that takes inspiration from the behavior of colonies of social insects and animal societies, such as flocks of birds or schools of fish (Blum and Li 2008, 43). The word “swarm” is an appropriate word because it has special characteristics not found in related terms such as “group.” The three key special characteristics of a swarm are decentralized control, lack of synchronicity, and the simplicity and homogeneity of the swarm; additionally, the swarm’s algorithms run in an asynchronous and decentralized fashion (Beni 2004, 2).

This thesis explores the area of UGV autonomy, investigating the effects of kinematics inputs—such as movement behavior, swarm size, detection range—and engagement inputs—such as sensors and weapon range—with an assumption that the hardware and software capabilities requirements mentioned previously are met.

B. OBJECTIVES

This thesis aims at developing and testing swarming algorithms as applied to combat UGVs to execute a search and destroy mission in an urban environment. The search phase of the mission focuses on exploring a given area in order to find all potential threats. The track and engage phase focuses on eliminating these threats. Furthermore, the thesis addresses three algorithms—the Least Visited Cell (LVC) guidance, the Advanced Least Visited Cell (ALVC) guidance, and the Particle Swarm Optimization (PSO)—developed for both outdoor and indoor environments in an urban area.

C. RESEARCH QUESTIONS

In order to meet the thesis objectives, this thesis strives to answer three critical questions:

- Is the algorithm developed suitable for the swarm of UGVs to achieve its mission?
- What are the strengths and weaknesses of the algorithms used in this thesis?

- What are the factors that affect the UGV swarm’s ability to achieve its mission?

D. SCOPE

In order to gain a holistic view of and insights on the UGV swarm’s system, a simple functional analysis at the engineering conceptual design level was conducted using the systems engineering approach.

1. Functional Analysis and Allocation

A functional analysis on the UGV combat swarm was performed to determine what the system needs to do. This analysis is depicted in Figure 3 and explained in more detail in the subsequent paragraphs.

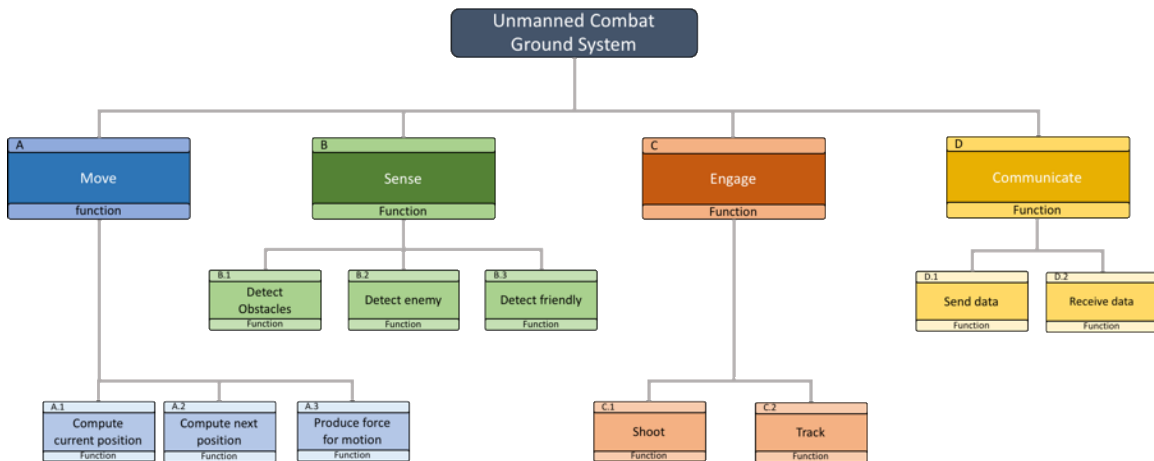


Figure 3. Functional decomposition of swarm combat UGV system.

(1) Move

First, the system requires the swarm of UGVs to move in the area of operation. In order to do that, the system needs the ability to compute the swarm’s current as well as its next position. It also needs to produce a force for motion in order to physically move itself.

(2) Sense

The system needs to be able to sense its surroundings. The sub-level of this function would be to detect obstacles so as not to collide with them, detect the enemy for engagement, and detect other UGVs for collision avoidance as well as for computation of the swarm's next position. An important sub-level of detecting the enemy is the ability to discern its status as dead or alive.

(3) Engage

A main purpose for the swarm UGV is to take down the enemy. To do that, each UGV within the swarm must be able to shoot. A sub-level of the shoot function includes computing range and aiming point. In the event that the shot failed, the swarm must be able to continue to pursue the enemy and continue to shoot.

(4) Communicate

For an algorithm such as the Particle Swarm Optimization (PSO) algorithm to work, the swarm must have the ability to communicate information such as its own position and the position of its target. In addition, information of visited locations would assist in an effective algorithm.

2. Functional Flow Block Diagram

The functional flow block diagram (FFBD) is shown in Figure 4. This process is categorized into two phases: search, and track and engage. The search phase focuses on the maneuvers of the swarm in order to find the enemy. The track and engage phase focuses on eliminating the enemy after the swarm has successfully found the enemy.

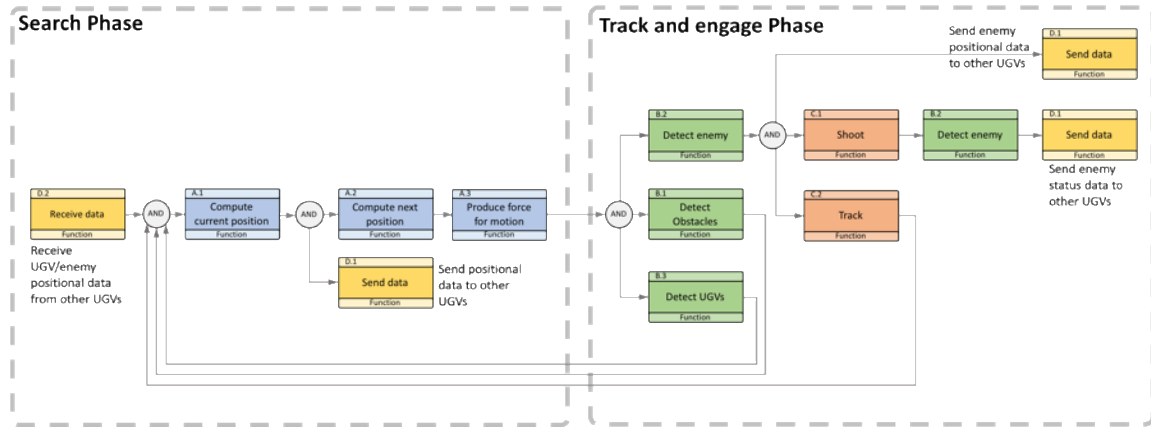


Figure 4. FFBD for swarm combat UGVs.

This thesis focuses on the creation and discussion of algorithms for both operation phases highlighted in the FFBD diagram in Figure 4 and offers insights on the input parameters from the research model. This thesis does not discuss hardware and software capabilities of the UGVs themselves.

E. ASSUMPTIONS

As such, the assumptions made for this thesis are as follows.

- The UGVs are able communicate with each other and will not experience any information delay or distortion.
- The UGVs are able to correctly identify obstacles, UGVs, and enemies all the time.
- There are no positional errors for UGVs and the enemy's location or positions.

Although these assumptions might not accurately reflect conditions in the real world, they allowed the author to simplify the model so as to gain insights into the algorithm built.

THIS PAGE INTENTIONALLY LEFT BLANK

II. LITERATURE REVIEW

This chapter details the review of literature from previous studies conducted by scholars and researchers. The chapter starts by reviewing various search algorithms for the search phase, followed by a consideration of the two-swarm optimization approach for the track and engage phase. Thereafter, existing applications of swarm algorithms and UGVs sensors and movement are discussed briefly.

A. SEARCH ALGORITHMS

Search algorithms are generally classified into exhaustive and heuristic search types. The exhaustive search algorithm explores all the possible options in the network during its execution to find the solution, and as such, it is time consuming. The heuristic search algorithm employs rules at every branching step and, in contrast to the exhaustive search algorithm, often includes some form of randomization to find the solution. In the case of coverage, exhaustive search guarantees complete coverage of the free space while the heuristic search approach does not. In short, heuristic approaches often trade accuracy for speed.

1. Exhaustive Search

A common exhaustive search is the classical exact cellular decomposition. This method breaks the examined space into strips called cells and proceeds to cover these cells via simple motions like “lawn mower” pattern (Galceran and Carreras 2013, 3). As shown in Figure 5, the space is broken down into six cells (vertical strips) and coverage would be complete after the robot finished its “lawn mower” pattern. Two popular cellular decomposition approaches that incorporate obstacle avoidance are discussed next.

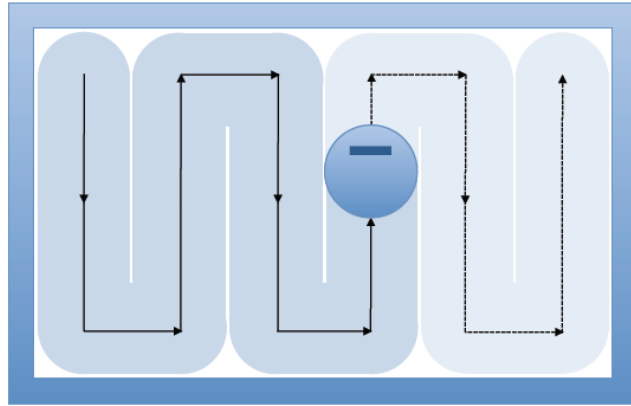


Figure 5. Robot moving in a “lawn mower” pattern through the cells.
 Source: Galceran and Carreras (2013).

a. *Trapezoidal Decomposition*

Galceran and Carreras (2013) mention that the cellular decomposition technique is simple and able to yield a complete solution. As shown in Figure 6, each cell is broken up into a trapezoid shape once the robot encounters an obstacle and merges the cell once the robot gets past the obstacle, and in this case, 12 cells are generated and complete coverage is achieved once all cells are visited (Galceran and Carreras 2013, 3). They also mention the drawback of this technique is that it requires many back and forth motions to achieve completeness as well as requires the obstacle to be polygonal.

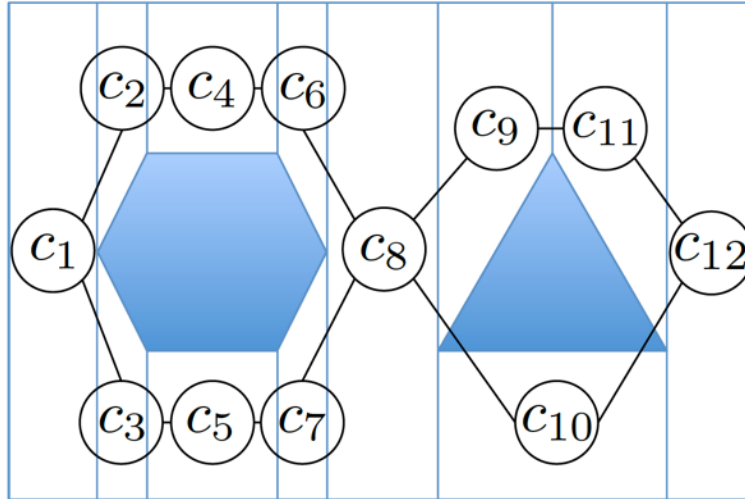


Figure 6. Broken up cells in the trapezoidal decomposition. Source: Galceran and Carreras (2013).

b. Boustrophedon Decomposition

Choset's (2000) work on boustrophedon cellular decomposition is an improvement on the trapezoidal cellular decomposition. Compared to the trapezoidal decomposition technique, his work is able to reduce the number of overlapping motions by setting critical points to mark the start and end of an obstacle and therefore reduce the number of cells. For example, for the case shown in Figure 6, the critical points would be at C1 and C8. Thus, C2, C4, and C6 and C3, C5, and C7 would be a single cell instead of three.

c. Grid-based Coverage using the Wavefront Algorithm

According to Galceran and Carreras (2013), "Grid-based methods use a representation of the environment decomposed into a collection of uniform grid cells" (13). They mention that grid cells are most commonly represented by a square; however, a different grid cell type, such as triangles or trapezoids, can also be used.

Zelinsky et al. (1993) work on grid-based coverage using the wavefront algorithm. This method assigns a specific number to each grid cell based on the distance between the start and goal cell, which is known. As seen in Figure 7, the nearest cells to the goal are assigned lower values, while the furthest cells are assigned higher values.

13	12	11	10	9	8	7	7	7	7	7	7	7	7
13	12	11	10	9	8	7	6	6	6	6	6	6	6
S	12	11	10	9	8	7	6	5	5	5	5	5	5
									4	4	4	4	4
9	8	7	6	5	4	3	3	3	3	3	3	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	1	G	1	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4

Figure 7. Assigned values for each cell using the wavefront algorithm.
Source: Zelinsky et al. (1993).

As shown in Figure 8, the path is created by selecting the unvisited neighboring cell with the highest value. A random decision would be made if there are two unvisited neighboring cells with same highest value.

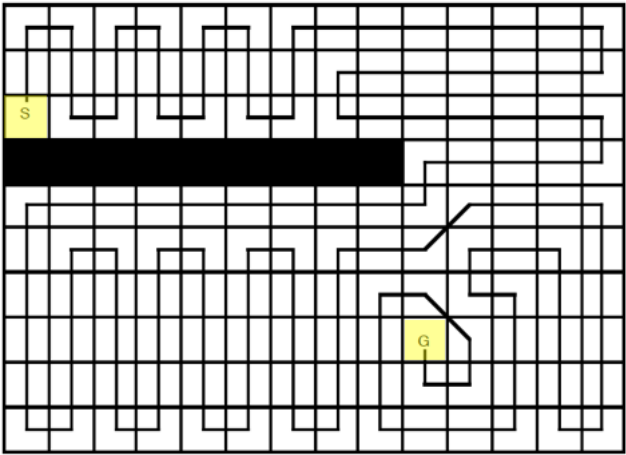


Figure 8. Path of complete coverage using wavefront algorithm. Source: Zelinsky et al. (1993).

Zelinsky et al. (1993) also presented a second distance transform generation using a new cost function instead of full coverage in order to find the shortest path. In this case,

the path is selected based on the lowest value instead of the highest. The advantage of the grid-based method for coverage is simplicity of implementation; however, this method suffers from memory issues as the environment gets larger and more complex (Galceran and Carreras 2013, 13).

2. Heuristic Search

The heuristic search algorithm is an approach that employs rules at every branching step and often includes some form of randomization in attempting to reach a solution. Two common heuristic search algorithms discussed in this section are the greedy and swarm algorithms.

a. Greedy Algorithm

A popular and well-known heuristic search is the greedy algorithm. Charlier's (1995) report on the greedy algorithms class states that the greedy algorithm must satisfy two conditions. One, the algorithm has to construct the solution step by step. Two, at each step, the best possible local choice is made. Its aim is to find a global optimum by performing a succession of local optimizations. In many cases, the greedy algorithm does not produce a global optimal solution. Nevertheless, a relatively approximate solution (locally optimal solution) could be found in a reasonably shorter period of time.

b. Swarm Algorithm

Blum and Li (2008) recognize Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) as the two notable swarm intelligence techniques for producing approximate solutions in a reasonable computation time period.

(1) Ant Colony Optimization

Dorigo et al. (1996) introduce ACO based on the behavior of ants in which they leave traces (pheromones) as they mark the route for their colony. The work of Goss et al. (1989) on the double bridge demonstrates an optimization method for finding the shortest path. Initially, the ants would explore both the long and the short bridges to a food source. Since the ants that took shorter bridge return to the colony faster than those on the longer

path, more pheromones are laid on the shorter path, hence encouraging the other members of the ant colony to use the shortest path. The limitation in this optimization method, as the paper noted, is that once a path is established, the introduction of a new bridge would not be explored due to the overwhelming number of pheromones existing in the original chosen path (Goss et al. 1989). This would prove to be a huge limitation in the context of this thesis as the objective (enemy) is constantly moving.

(2) Particle Swarm Optimization

Kennedy and Eberhart's (1995) work on particle swarm optimization (PSO) is a popular algorithm for swarm research. The two main component methodologies for this work correspond to artificial life, such as bird-flocking or fish-schooling, as well as work on genetic algorithms and evolutionary programming (Kennedy and Eberhart 1995). To explain the essence of the PSO algorithm, let us assume that each agent evaluates its current distance from $\{ x, y \} = [100, 100]$ point. An important factor to note is that the agents require an objective in order to be able to evaluate their position.

$$Eval = \sqrt{(x - 100)^2} + \sqrt{(y - 100)^2} \quad (1)$$

The PSO concept is then to change the agent's velocity (V_i) at every time step towards its personal best position (Pbest) and global best position (Gbest).

$$V_i = V_{i-1} + [Acceleration \cdot (Rand\ of\ 0\ to\ 1) \cdot (Pbest - Current\ position)] \\ + [Acceleration \cdot (Rand\ of\ 0\ to\ 1) \cdot (Gbest - Current\ position)] \quad (2)$$

In Equation (2), Pbest is defined as the closest position from the goal that a particular agent has been and Gbest is the closest position from the goal that any agent has been. At any one time, there would be an agent whose Pbest is the Gbest. Acceleration represents the weight that pulls each agent towards the Pbest and Gbest.

The research of Shi and Eberhart (1998) improves the PSO algorithm by introducing inertia weight (I) to act as a constraint to control the global exploration ability of an agent.

$$V_i = I \cdot V_{i-1} + [Acceleration \cdot (Rand \text{ of } 0 \text{ to } 1) \cdot (Pbest - Current \ position)] + [Acceleration \cdot (Rand \text{ of } 0 \text{ to } 1) \cdot (Gbest - Current \ position)] \quad (3)$$

The advantages of PSO are that it is simple, having very few parameters to adjust. It is effective and works well in a wide variety of applications. Thus, PSO seems to be a viable option in the context of this thesis for a swarm of UGVs moving towards an enemy once it is located.

3. Summary of Search and Swarm Optimization Algorithms Used in this Research

Table 2 shows a summary of the search algorithms discussed previously.

Table 2. Summary of search algorithms

Category	Approach	Advantages	Disadvantages
Exhaustive Search	Trapezoidal Decomposition	Simple and easy to implement.	Requires knowledge of environment.
	Boustrophedon Decomposition	Reduced processing time compared	Might not work well in dense environment with complex structures.
	Grid-based Wavefront Algorithm	Simple and easy to implement.	Requires knowledge of environment. Suffer from exponential growth of memory usage as environment becomes complex.
Heuristic Search	Greedy Algorithm	Simple and easy to implement.	Might not find global optimal solution.
Heuristic Search (Swarm Optimization)	Ant Colony Optimization	Able to find local optimal solution in a short time.	Poor ability to adapt or change route. Does not work for moving objective.
	Particle Swarm Optimization	Able to find local optimal solution in a short time. Works for moving objective.	Easy to fall into local optimal solution.

For the purposes of this thesis focusing on a search and destroy mission, the complexity of finding a moving target mitigates the main disadvantage of a heuristic search not being able to locate the global optimal solution. Thus, the heuristic search approach might yield better results in a shorter time. Furthermore, full knowledge of the environment in many scenarios, especially for this research, is unrealistic. Even if a blueprint of the area of operations is available, it cannot depict all the potential obstacles, natural or man-made, in any urban environment. As such, the exhaustive search algorithm is not suitable.

Although the Grid-based Wavefront algorithm is not suitable either, as shown previously, this thesis does employ a grid-based concept for the foundation of the developed algorithm because of its potential and simplicity. For the search phase, the greedy algorithm based on a grid concept is employed.

For the track and engage phase, this thesis employs the PSO algorithm instead of the ACO algorithm. ACO was rejected because of the algorithm's lack of flexibility to change objectives once it establishes a path. However, the concept of pheromones from the ACO algorithm is employed together with the greedy algorithm in the search phase in order to explore the least visited areas.

B. APPLICATIONS OF SWARM ALGORITHMS

Quite a few research studies have been conducted to develop swarming algorithms and assess their effectiveness in a variety of possible real-world applications. Among these studies, the PSO algorithm emerged as one of the simplest and yet effective ones. For example, the implementation of PSO to fine-tune a profile-matching algorithm to learn users' preferences and make suggestions in E-commerce was studied (Ujjin and Bentley 2003). This research concluded that in the majority of cases, the PSO system obtained better prediction accuracy than non-adaptive approaches such as genetic and Pearson's algorithms.

In another study, a PSO-based image clustering method was developed and compared with K-means, fuzzy C-means, K-harmonic means, and genetic algorithm approaches (Omran et al. 2005). Experimental results provided in this study showed that

the PSO image classifier produced better results than the other image classifiers for all measured criteria.

In a similar domain, another swarm algorithm, Ant Colony Clustering, was employed to discover Web usage patterns. The empirical results demonstrated that the ACO algorithm performed well compared to a self-organizing map neural network (Abraham and Ramos 2003). Swarm algorithms have also been used for forecasting. For example, the ACO approach was used to estimate energy demand (Toksari 2007). For the same problem, a combined ACO-PSO model was also developed (Ünler 2008).

C. MACHINE VISION

In order for a UGV to perform its tasks, it requires the ability to sense and gain situational awareness through the inputs of its sensors. Vision, among all the other senses, undoubtedly provides the most data and is most appropriate in this context. According to the unmanned ground systems roadmap report by the Robotic Systems Joint Project Office of the DoD (2011), the image sensors for UGVs currently operate in three spectrums; visible, near infrared, and thermal infrared (Department of Defense 2011, 28). This roadmap reports that from 2009 to 2011, significant improvements in technology related to image sensors and vision capabilities have been fostered by research and development within both the United States government and industry in various critical areas:

- Demonstrated obstacle detection and avoidance, visual odometry, lane detection, and sensor fusion
- Investigated stereoscopic vision and terrain classification technologies
- Matured vision-based navigation and learning technologies
- Matured vision technologies that enable UGVs to safely operate within urban environments among humans, animals, and vehicles (U.S. Department of Defense 2011, 29)

This roadmap also describes future developments the DoD plans to pursue. In the short term, it aims to improve imaging in order to increase the number of pixels for more image detail or wider field of view to include 360-degree images. Since the publication of

the 2011 roadmap, not only have 360-degree field of view cameras been developed, they have become so affordable that they have been launched as consumer products. An article written by Goldman (2016) on CNET reviews ten of these cameras and their prices for consumers in 2016, which signals the prospect of rapid success in technological advancements required for the roadmap's proposed goals.

D. HOLONOMIC BEHAVIOR

In robotics, holonomic drive refers to the relationship between controllable and total degrees of freedom of a vehicle. If the controllable degree of freedom is equal to total degrees of freedom, then the robot is said to be holonomic (Robot Platform n.d.). According to Morin and Samson (2004), the control of non-holonomic vehicles is a very active research field because automated wheeled vehicles are now envisioned for use in daily life and the military.

Naffin and Sukhatme's (2004) work studies the problem of assembling and maintaining formations of robot. Their approach was to dynamically create a formation from wandering individual robots by establishing and negotiating protocols and rules when the robots encounter each other. The four objective formations of choice for their study were the column, line, wedge, and diamond, and the three metrics used to determine performance were positional error, duration of time required to form the required formation, and duration of time the formation could be maintained. Their result shows that compared to robots with non-holonomic drives, robots equipped with holonomic drives were able to get together in three out of the four formations in the study. This field of work is related to this thesis topic in terms of UGV motion and the insights gained from the effects of non-holonomic and holonomic drive, and can act as a good data reference for this thesis.

III. MODELING

This chapter discusses several aspects of urban environment engagement modeling starting from modeling motion primitives in the open-space obstacle-free environment and gradually proceeding to the specifics of the track and engage phase modeling.

Hereinafter, the UGV swarm executing the mission is referred to as the Blue forces, while the threats are referred to as the Red forces. Simulations are bounded by the following assumptions:

- Blue agents are able to communicate with each other without any information delay or distortion.
- Blue agents are able to correctly identify obstacles, other Blue and Red agents.
- The starting position for the Blue forces is fixed to several feasible options, dependent on the actual urban environment.
- Errors in estimating agents' positions are negligible.
- Blue agents are in the offensive mode while Red agents are in defensive posture.

A. MOTION PRIMITIVES

Without loss of generality, agent motion is considered to be conducted along the edges of the grid. In all simulations of this paper a square 100-by-100 cell grid was used (actual grid size depends on the area to cover and agents' mobility). With $\mathbf{P}_{ij} = [x_{ij}, y_{ij}]^T$ representing the current (ith iteration) two-dimensional cell position of the jth agent, their kinematics are defined as

$$\mathbf{P}_{i,j+1} = \mathbf{P}_{ij} + \Delta\mathbf{P}_{ij} \quad (4)$$

The assumptions are that

- distance traveled per iteration for each agent is limited to its eight surrounding cells so that the change in x or y coordinates is either 1 or 0 (Figure 9);
- each agent is able to broadcast and receive its coordinates as well as keep track of the previously visited cells (maintain visitation map); and
- simulation scoring is based on the status of Blue and Red agents (either dead or alive).

At each iteration, every Blue agent determines its next position by evaluating the immediate surrounding cells (Cells 1 to 8), as shown in Figure 9, and randomly picking one of them.

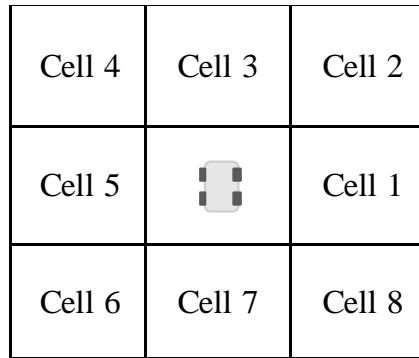


Figure 9. Agent surrounding cells.

The velocity vector is then computed based on the change in the x and y coordinates

$$\mathbf{V}_{i+1,j} = k_t \Delta \mathbf{P}_{ij} \quad (5)$$

where k_t is some scaling factor that can be used to account for the physical size of the cell and number of interactions representing one second ($k_t = 1$ means that velocity components are expressed in cells per iteration).

Compared to the Blue agents pursuing an exploration or elimination objective (based on the phase of engagement) and therefore moving around, the Red agents are in a defensive posture and are likely not to move around too much. Hence, while maintaining the same kinematics, the Red agents' next position is determined randomly among its surrounding obstacle-free cells.

B. MOTION CONSTRAINTS

In this chapter, the motion constraints of two guidance algorithm, Least Visited Cell Guidance and Advance Least Visited Cell guidance, that was developed for this thesis would be discussed.

1. Least Visited Cell Guidance

To achieve the maximum coverage of the area of operations while searching for Red agents, the Blue agents employ the Least Visited Cell (LVC) guidance. This guidance dictates for each Blue agent to access eight surrounding cells, identifying those that have been visited by itself or other agents. Each Blue agent then randomly selects one of the unvisited cells as a next move. For example, in the situation shown in Figure 10, any one of Cells 2, 3, 6, or 8 would randomly be selected as a Blue agent's next designation. In the event that all surrounding cells have been visited already, the agent randomly selects one of them.


Cell 4 (Visited)	Cell 3	Cell 2
Cell 5 (Visited)		Cell 1 (Visited)
Cell 6	Cell 7 (Visited)	Cell 8

Figure 10. Agent's surrounding visited and unvisited cells.

a. Collision Avoidance

Collision avoidance is modeled in such a manner that a UGV agent cannot move to a cell that is occupied by other agents. The cell that is occupied by other agents would not be considered as a possible designation. The logic for the remaining cells would follow the sequence described in the preceding paragraph. For example, in Figure 11, either Cell 2 or Cell 6 would randomly be selected as the agent’s next designation. In an event that all cells are occupied, the agent would remain in its current position until a cell is unoccupied.




Cell 4 (Visited)	Cell 3 	Cell 2
Cell 5 (Visited)		Cell 1 (Visited)
Cell 6	Cell 7 (Visited)	Cell 8 

Figure 11. Surroundings visited, unvisited, and occupied cells.

b. Non-Holonomicity Constraint

Another constraint is imposed to represent a particular vehicle dynamics. While some UGVs may be holonomic, meaning that they can move in any direction at any instance of time, other UGVs are not holonomic and have certain turn rate limitations. The choice of a particular dynamic is modeled using a non-holonomicity constraint limiting the direction of the agent’s next move relative to the direction of the previous move. For example, Figure 12 shows a situation of imposing a 90-degree non-holonomicity constraint limiting the UGV’s next move to just three cells (corresponding to the maximum turn rate of ± 45 degrees per iteration). In this particular situation, the agent would pick Cell 4 as its destination. In the event where Cells 2, 3, and 4 are visited, the agent would randomly select among them.

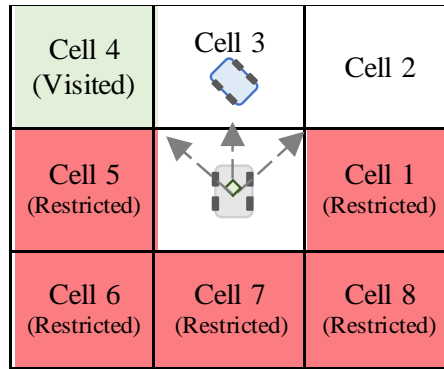


Figure 12. Surrounding cells with a non-holonomicity constraint of 90 degrees.

c. Obstacles

Similar to collision avoidance, obstacles such as boundaries or building walls are programmed as occupied cells and the UGV agent would not be able to move to those cells. For example, in Figure 13, the agent would select Cell 4 as its designation as Cells 1, 2, and 8 are the map boundaries; Cell 3 is occupied; and Cells 5, 6, and 7 are restricted because of the non-holonomicity constraint of 90 degrees.

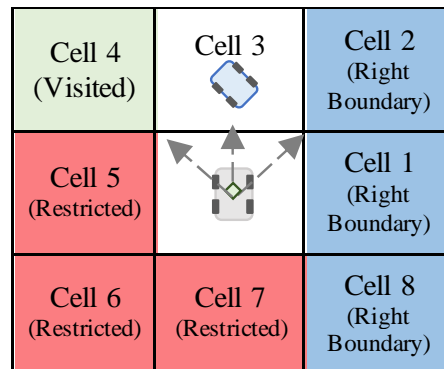


Figure 13. Surrounding cells with a non-holonomicity constraint of 90 degrees and obstacles.

2. Advanced Least Visited Cell Guidance

Advanced Least Visited Cell (ALVC) guidance is a modification of LVC guidance aimed at achieving better performance. According to ALVC guidance when all the

surrounding cells are visited, instead of randomly choosing a cell as its designation, the UGV agent extends its search scope beyond its immediate surrounding cells until it finds an unvisited cell. Its designation among the eight cells would be prioritized based on the direction of the nearest unvisited cell identified. Figure 14 shows two illustrations of a UGV agent with a non-holonomicity constraint of 90 degrees. In both illustrations, after the closest unvisited cell is identified, Cell 5 is given priority 1, followed by Cells 4 and 6 assigned priority 2, Cells 3 and 7 assigned priority 3, followed by Cells 2 and 8 assigned priority 4. Finally, the last priority is assigned to the cell in the opposite direction from the target. To illustrate, on the left side of Figure 14, as there are no obstacles or other agents in the surrounding cells, the UGV agent would select either Cell 4 or Cell 6 (priority 2) as its designation as Cell 5 (priority 1) cannot be accessed due to the non-holonomicity constraint. In the example on the right side of Figure 14, the UGV agent would select Cell 3 (priority 3) as its designation because all priority 1 and 2 designations are restricted. The entire process repeats itself once 100 percent area coverage is achieved.

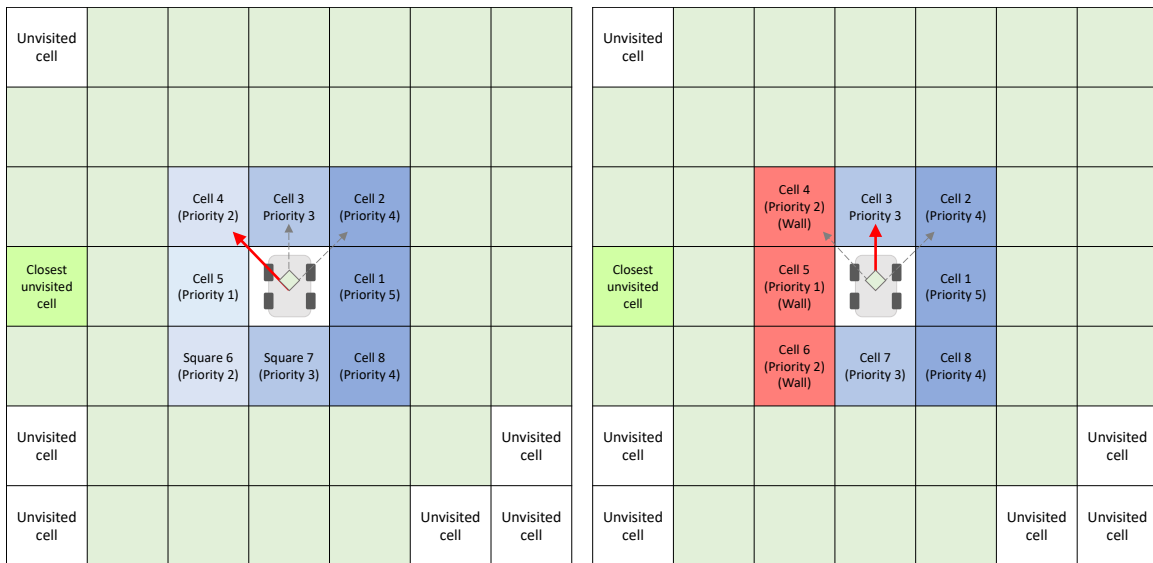


Figure 14. Illustrations of improved algorithm with non-holonomicity constraint of 90 degrees.

C. PARTICLE SWARM OPTIMIZATION

Once the track and engage phase of the mission is triggered, the Blue forces exercise the Particle Swarm Optimization (PSO) algorithm. In this case, the Blue agents' velocity vectors are computed according to Equation (6) (Shi and Eberhart 1998):

$$\mathbf{V}_{i+1,j} = w_I \mathbf{V}_{ij} + w_P \text{rand}_{ij} (\mathbf{P}_{ij}^{best} - \mathbf{P}_{ij}) + w_G \text{Rand}_{ij} (\mathbf{G}_i^{best} - \mathbf{P}_{ij}) \quad (6)$$

In this equation, the first term on the right-hand side is responsible for global search ability with the “inertia” weight $w_I = 1$, the second (cognition) term represents the private thinking of each agent trying to steer towards an individual best position from the past, \mathbf{P}_{ij}^{best} , and the third (social) term represents collaboration among all agents accommodating the knowledge of the best global position with respect to the detected Red agent, \mathbf{G}_i^{best} (where \mathbf{G}_i^{best} is one of the position vectors \mathbf{P}_{ij}^{best} closest to the detected Red agent). The Rand_{ij} and rand_{ij} are two random generators in the range [0;1] and the weighting coefficients w_P and w_G are chosen to be 2 to make the average weight of the second and third terms to be 1 (Shi and Eberhart 1998).

Once the track and engage phase is triggered, the values of \mathbf{P}_{ij}^{best} are chosen between the current and previous position by evaluating their distance from the Red agent that was detected. The cell position vector for the jth agent is computed as:

$$\mathbf{P}_{i,j+1} = \mathbf{P}_{ij} + \mathbf{V}_{i+1,j} \quad (7)$$

The PSO guidance goal is to steer the swarm towards a detected Red agent to increase the overall probability of kill, while still searching the area for other Red agents.

In an event that two or more Blue agents encounter different Red agents and these Blue agents are equidistant from their detected Red agents, there will be two \mathbf{G}_i^{best} positions, hence two goals. In this case each swarm agent adopts the \mathbf{G}_i^{best} of the Blue agent closest to it. This splits the swarm allowing it to pursue two or more Red agents at the same time, as seen in the example shown in Figure 15. Muhammad Raza (2018) wrote a MATLAB script for particle swarm optimization that is referenced in this thesis.

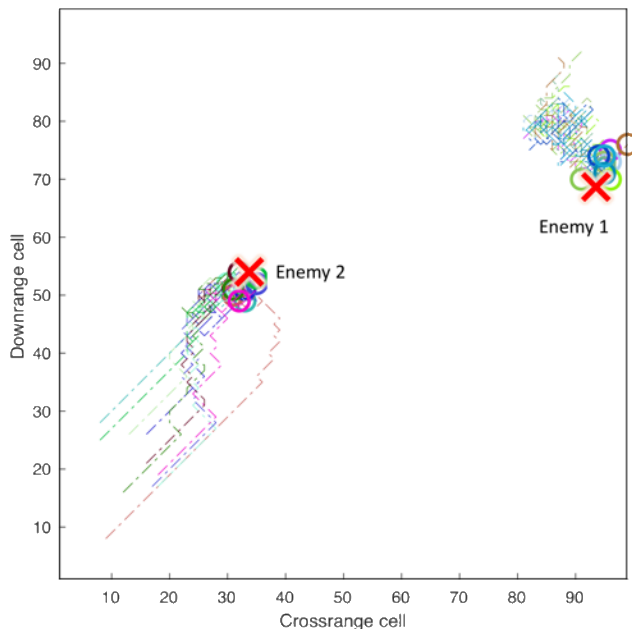


Figure 15. Example of a swarm pursuing two Red agents.

During the track and engage phase (PSO guidance), the algorithm can switch back to the search phase (LVC guidance) in the case of one of the following three events:

- The Blue agents lose track of the Red agent(s) because of the Red agent's maneuvers.
- The Blue agent tracking the Red agent is eliminated (killed).
- The Red agent that is being tracked is eliminated.

D. ENGAGEMENT RULES

Engagement between the Blue and Red agents is modeled as a probability event defined by five varied parameters:

- Detection range, $d_d^{B \rightarrow R}$
- Engagement range, $d_k^{B \rightarrow R}$ and $d_k^{R \rightarrow B}$, respectively
- Offensive capability, $P_k^{B \rightarrow R}$ and $P_k^{R \rightarrow B}$, respectively

The nominal values are $d_d^{B \rightarrow R} = 5 \text{ cells}$, $d_k^{B \rightarrow R} = 1 \text{ cell}$, $d_k^{R \rightarrow B} = 2 \text{ cells}$, and $P_k^{B \rightarrow R} = P_k^{R \rightarrow B} = 0.1$. If one of the Blue agents comes closer than $d_d^{B \rightarrow R}$ to any Red agent, the track and engage phase of the mission is triggered.

At each iteration when agents are within their respective engagement range, a random number from zero to one is generated. If this number happens to be less than the corresponding probability of kill, it is considered as a successful kill.

The Red agent always has a priority to shoot first. The reason for the Red forces to strike first is that in the considered scenario the Red forces are in a defensive position and likely to spot the Blue agent first. The difference in the engagement distance is caused by the same consideration.

E. OPERATIONAL ENVIRONMENT

The modeling aspect of the three operational environments, open space, outdoor as well as indoor urban environment would be discussed in this section.

a. Open Space Environment

The open space environment is a 100-by-100 cell operational area that does not consist of any obstacles. It is used to study the effects of various parameters without the interference of obstacles.

b. Mapping of Outdoor Urban Operational Area

An urban facility named, the impossible city, in Monterey, California, was chosen as an operational scenario and modeled to verify and evaluate the developed algorithm as well as the findings made in the early sections of this thesis. Figure 16 shows the model built in MATLAB in comparison with Google satellite images. While green boxes represent vegetation areas and blue boxes are buildings, the UGV and the enemy recognize both as obstacles and do not differentiate between them.

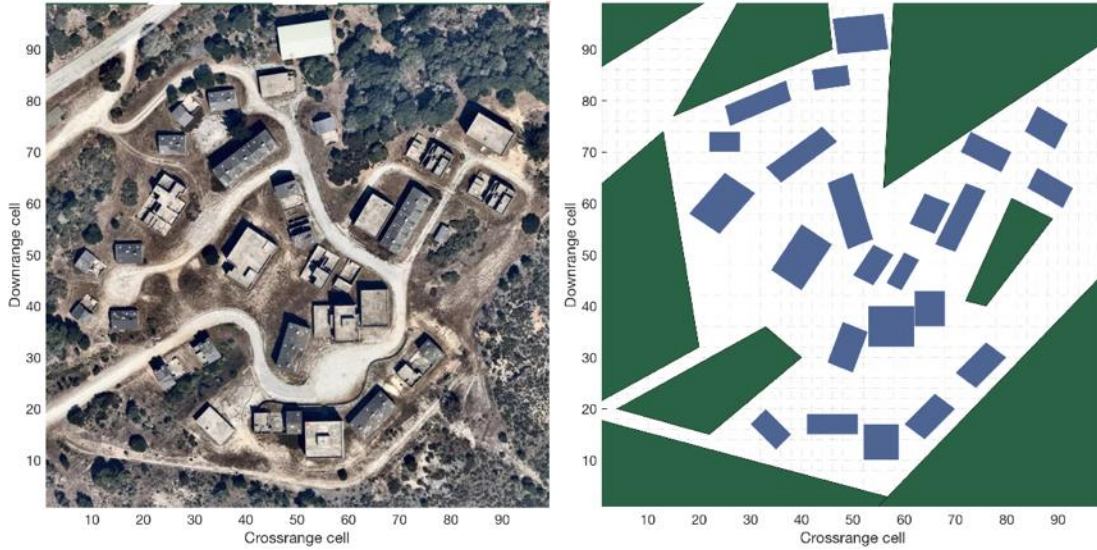


Figure 16. Model simulation of Impossible City at Fort Ord, California (right) and Google map view (left).

c. Indoor Operational Area

Indoor search is inevitable in any urban operation. Figure 17 shows an example of an indoor floorplan that was built to verify and evaluate the developed algorithm.

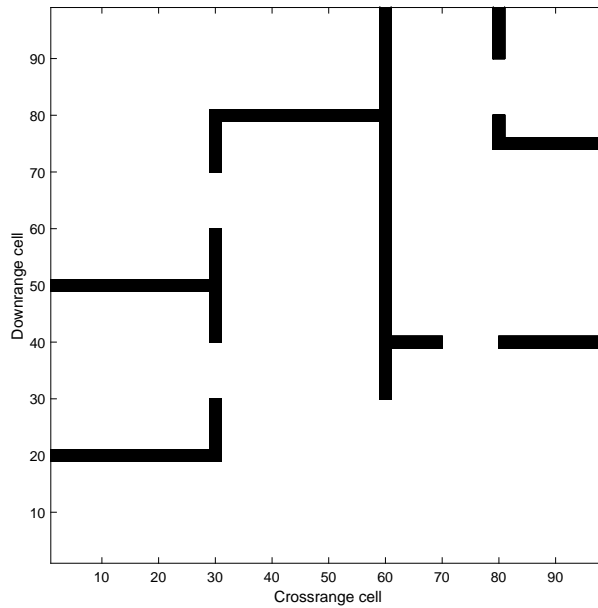


Figure 17. Indoor floorplan of a room.

IV. SEARCH PHASE STUDY

This chapter explores the developed search-phase guidance based on the LVC algorithm. The goal of this phase is to provide full coverage of the operational area. First, this section shows the effects of the swarm size, number of iterations, and starting configuration while operating in the obstacle-free environment. Then, the real-world constraints are added. The discussion ends with examples of full-scale simulations in the outdoor and indoor urban environments.

A. EFFECT OF SWARM SIZE

The effect of a swarm size with a fixed number of iterations was investigated first. To this end, Figure 18 shows the error plot of varying swarm size for 1,000 iterations based on 30 runs. It demonstrates an obvious result that with a fixed number of iterations or, in other words, within the same fixed time frame, having more agents leads to more thorough coverage of a given area following the logarithmic law.

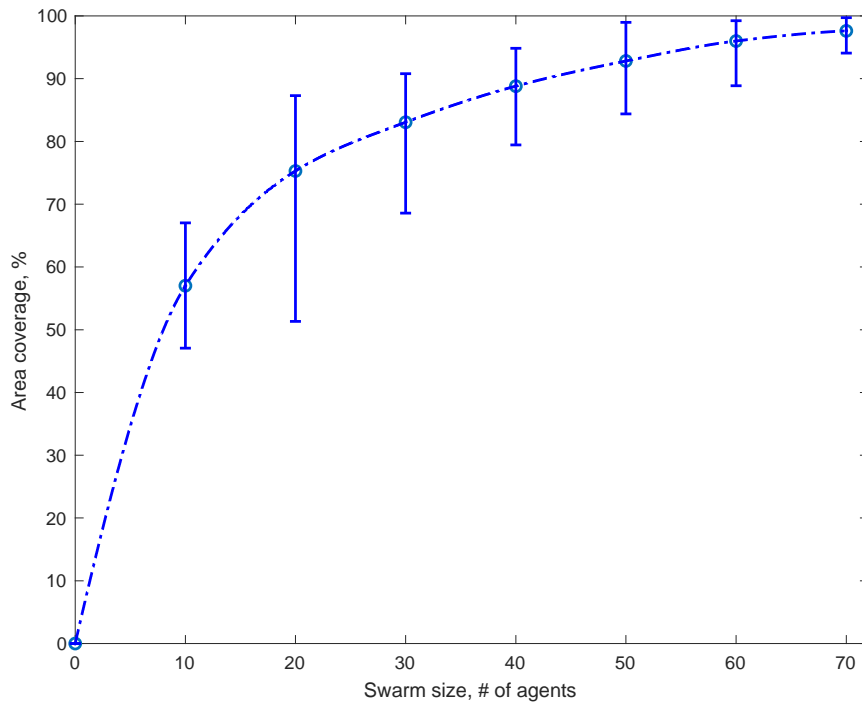


Figure 18. Effect of swarm size on area coverage.

As swarm size reaches beyond its saturation point, further increase of swarm size yields a diminishing return of area coverage. Thus, it would be ineffective to achieve maximum coverage purely by increasing the swarm size.

Figure 19 shows the 3D mesh plot for a swarm size of 10 agents with 55 percent area coverage while Figure 20 shows the 3D mesh plot for a swarm size of 70 agents with 94.8 percent area coverage.

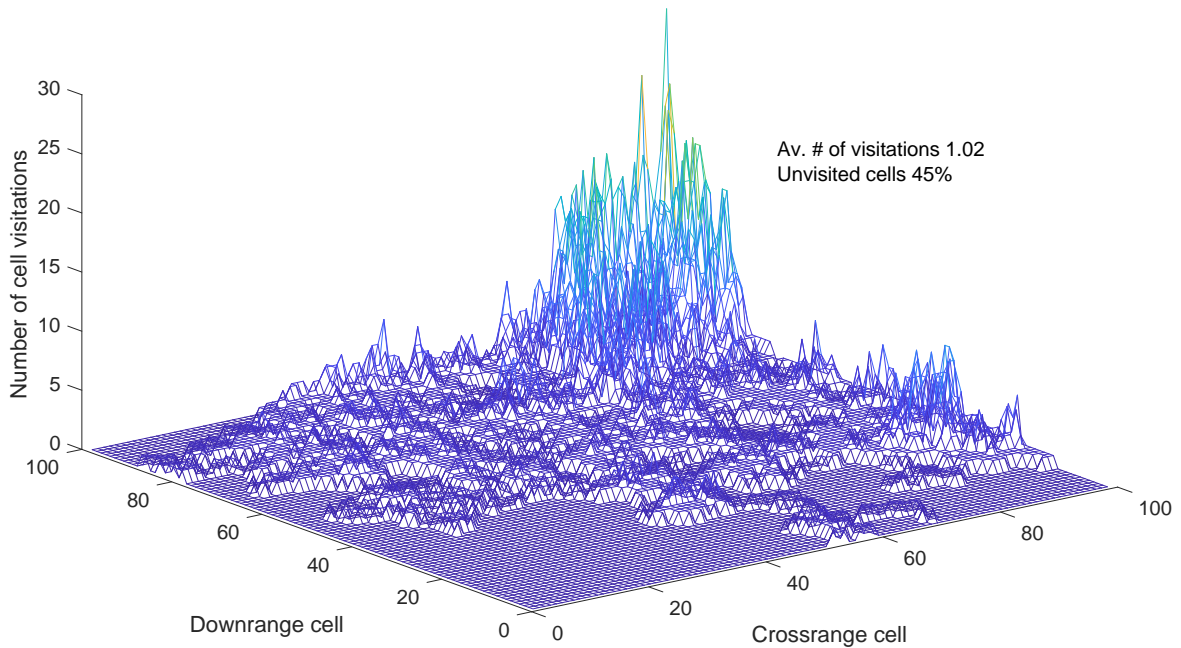


Figure 19. Average number of visits and number of cells for a swarm size of 10 agents with 1,000 iterations.

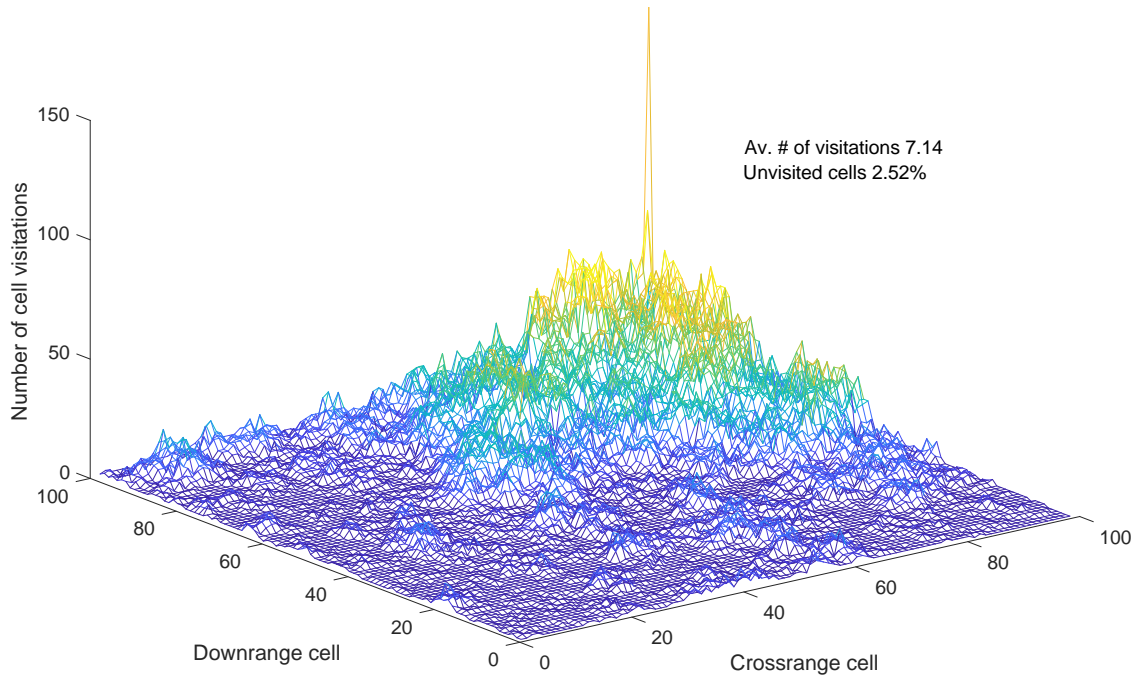


Figure 20. Average number of visits and number of cells for a swarm size of 70 agents with 1,000 iterations.

All Blue agents enter the area from the furthest entry point, then spread around the area. That is why the number of visits at the entry point shows a spike.

B. AREA COVERAGE VERSUS THE NUMBER OF ITERATIONS

The effect of limiting the maximum number of iterations is shown in Figure 21. In this specific case, simulations were conducted with a fixed swarm size of 20 agents.

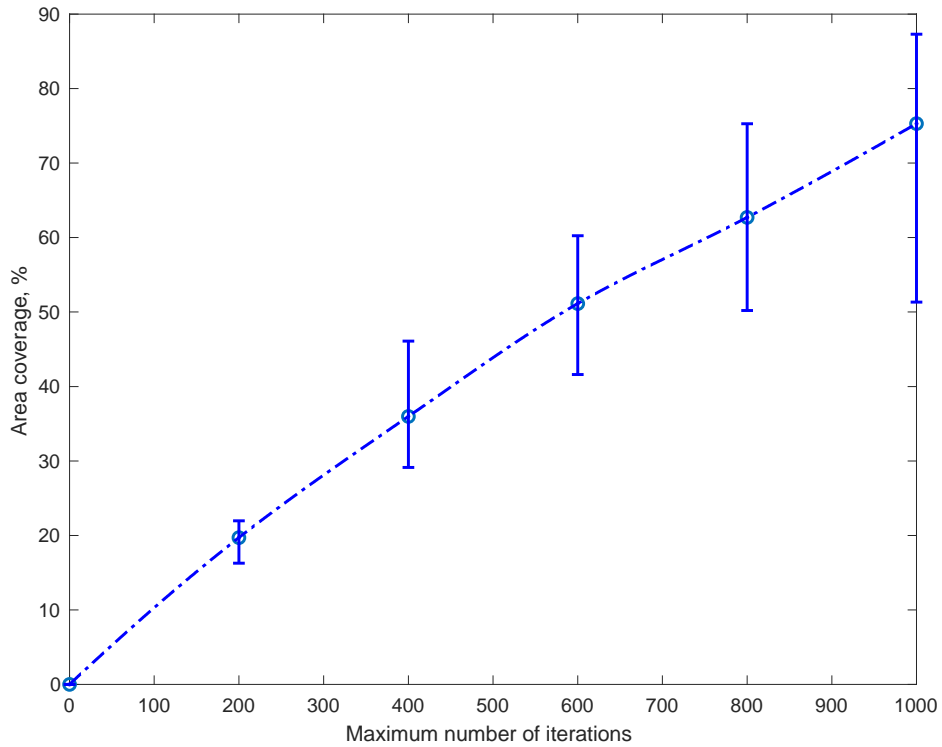


Figure 21. Effects of the maximum number of iterations on area coverage.

As seen in Figure 21, using the maximum number of iterations seems to have a linear effect on coverage. As expected, more iterations result in a fuller area coverage. It is also observed, however, that the variance increases with an increase in the number of iterations. This is likely due to the increased number of possible solutions as more iterations are performed, therefore increasing uncertainty in area coverage.

Figure 22 shows the trajectory plot of 19 percent area coverage on 200 iterations, and Figure 23 shows the trajectory plot of 70 percent area coverage on 1,000 iterations. Both simulations have a swarm size of 20 agents, where different colors in the plot correspond to different agents.

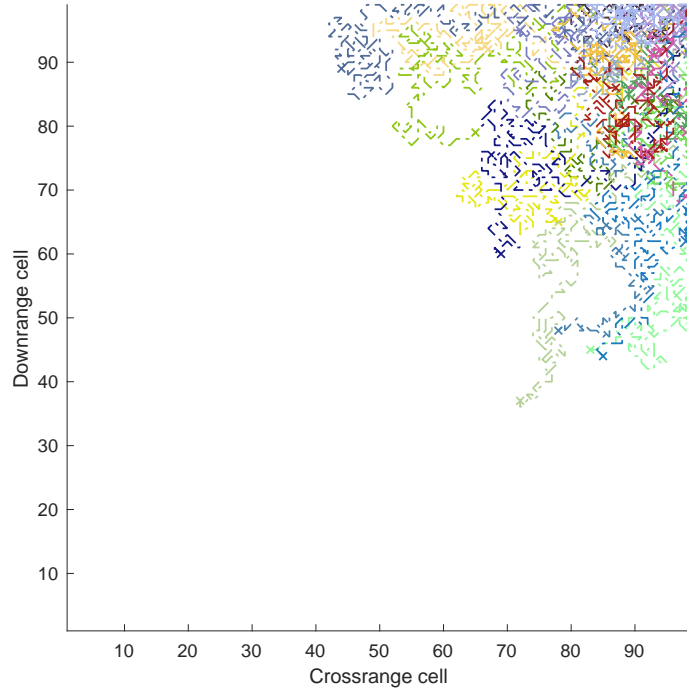


Figure 22. Trajectory plot of 19 percent coverage for 20 agents on 200 iterations.

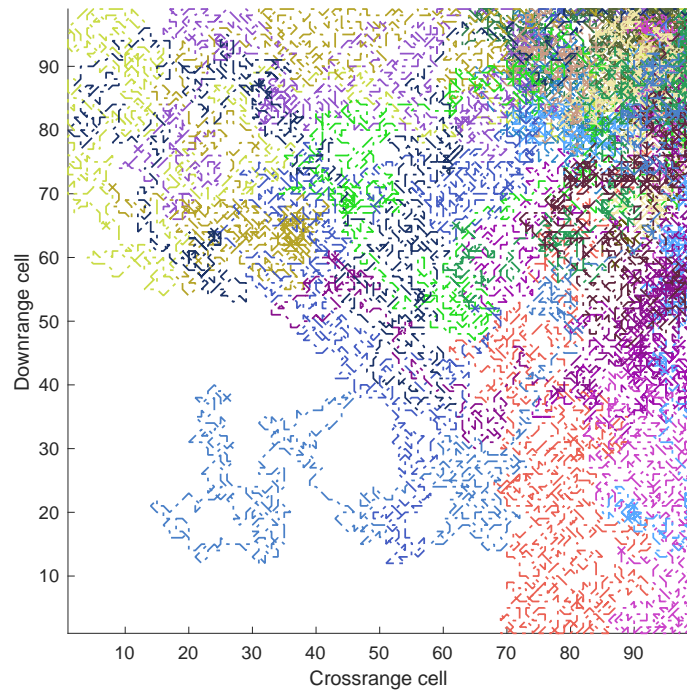


Figure 23. Trajectory plot of 70 percent coverage for 20 agents on 1,000 iterations.

C. SWARM SIZE VERSUS NUMBER OF ITERATIONS

Combining the results from the simulations in the previous sections allows investigation of the effect of varying both the maximum number of iterations and the swarm size simultaneously. Figure 24 shows the effect of iteration and swarm size on area coverage. This figure also shows the net effect of increasing the number of iterations for the different-size swarm.

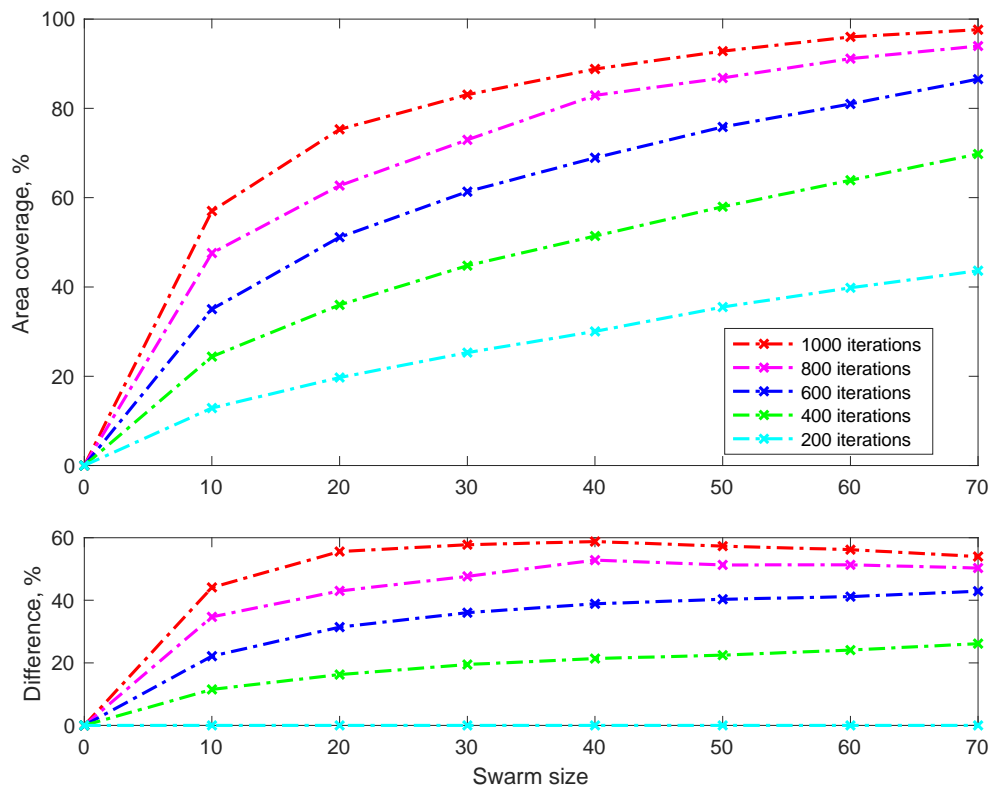


Figure 24. Effects of swarm size on area coverage with iteration comparison.

The increase in iterations from 200 to 400 yields the best improvement at 20 percent. By contrast, there is a diminishing return as the number of iterations is increased for a particular swarm size, as discussed in the previous section. As can be seen, achieving 100 percent area coverage is not as feasible or effective due to saturation of both swarm size and number of iterations.

Users of the developed system would likely be more interested in the required swarm size or number of iterations needed to achieve a pre-determined amount of area coverage. Figure 25 shows a chart that provides an estimate of swarm size or number of iterations needed given a required percentage of area coverage.

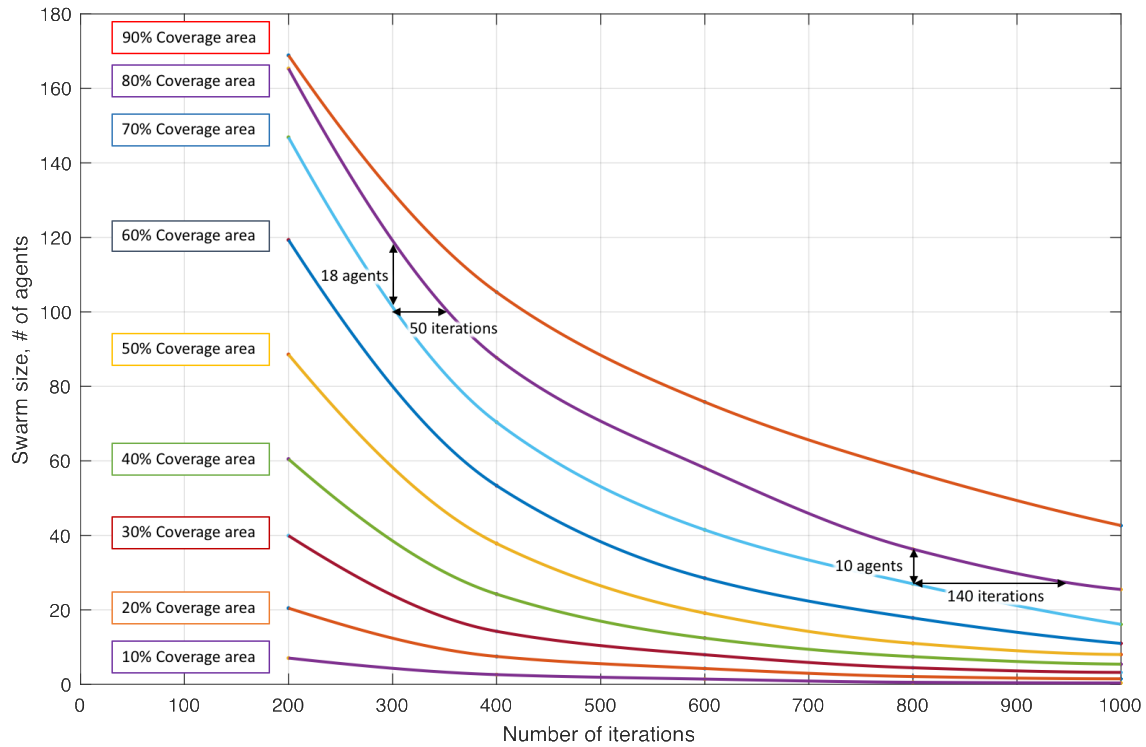


Figure 25. Estimated swarm size or number of iterations needed to achieve required area coverage.

Figure 25 shows a couple of points exhibiting a different relative improvement along the both coordinates needed to achieve a higher area coverage value. Particularly, the improvement in area coverage from 70 percent to 80 percent (i.e., a 10 percent improvement) requires an 18 percent to 34 percent increase in the number of agents within the 300 to 800 iterations range, or an 17 percent increase of the number of iterations for a 30- and 100-agent swarms.

D. EFFECT OF STARTING CONFIGURATION

Depending on the actual conditions, multiple entry points may be available. For example, an urban environment might feature several roads leading to the center of a village. The UGV swarm could also be delivered from an aerial platform. For the indoor engagement, multiple doors and windows could be used. As such, previous simulations were repeated for several representative starting configurations, which are depicted in Figure 26 through 31.

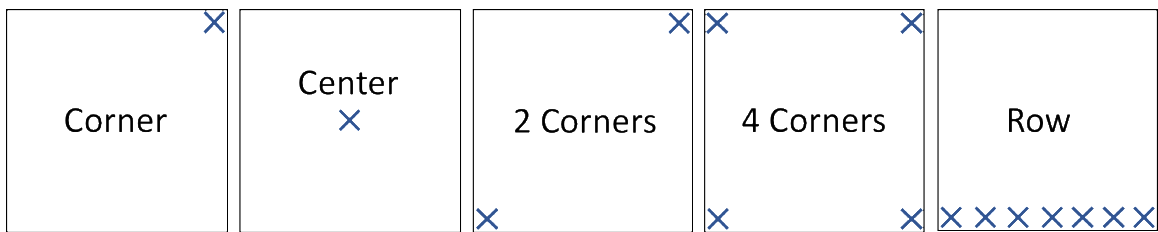


Figure 26. Various starting configurations.

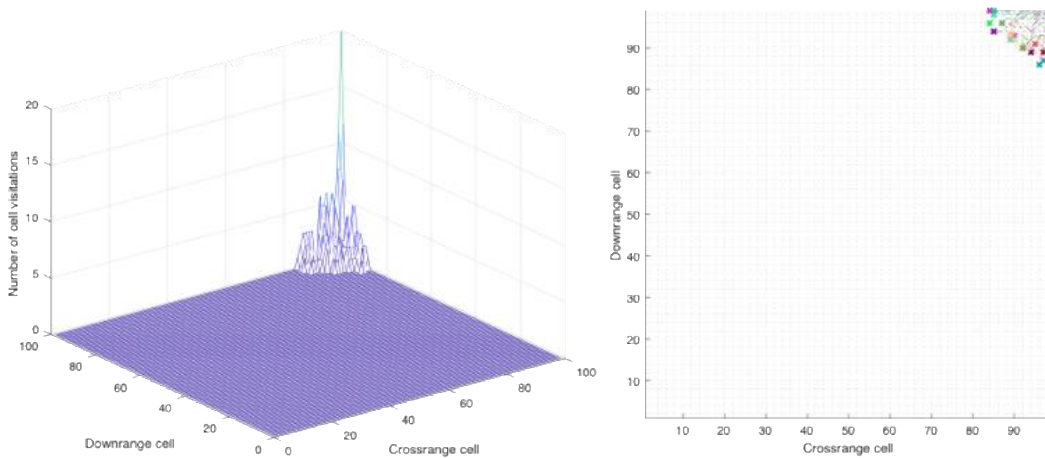


Figure 27. Snapshot of Corner starting configuration.

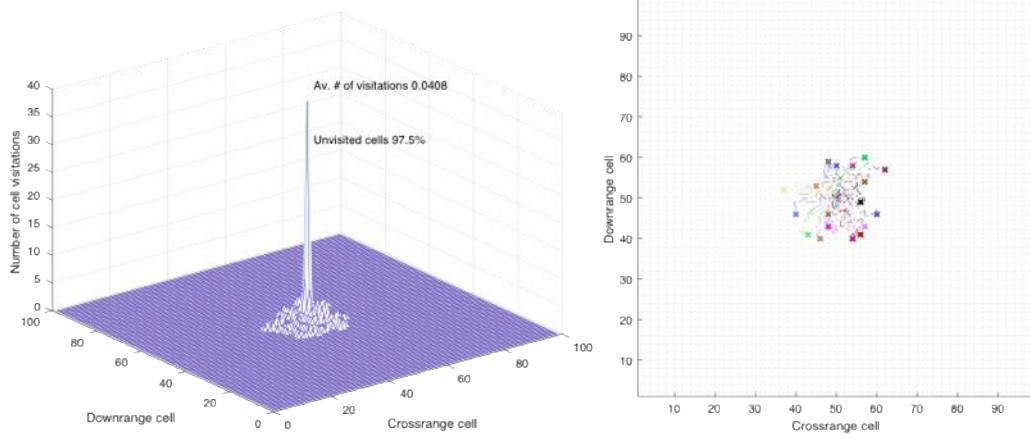


Figure 28. Snapshot of Center starting configuration.

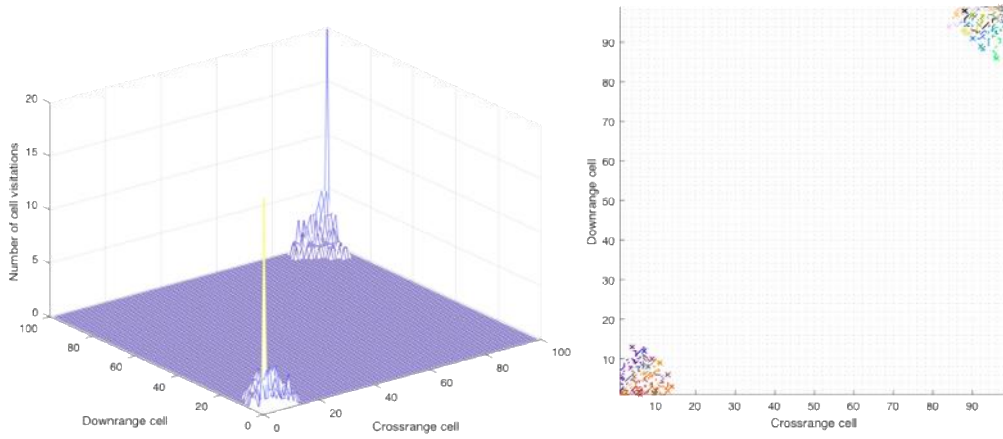


Figure 29. Snapshot of Two-Corners starting configuration.

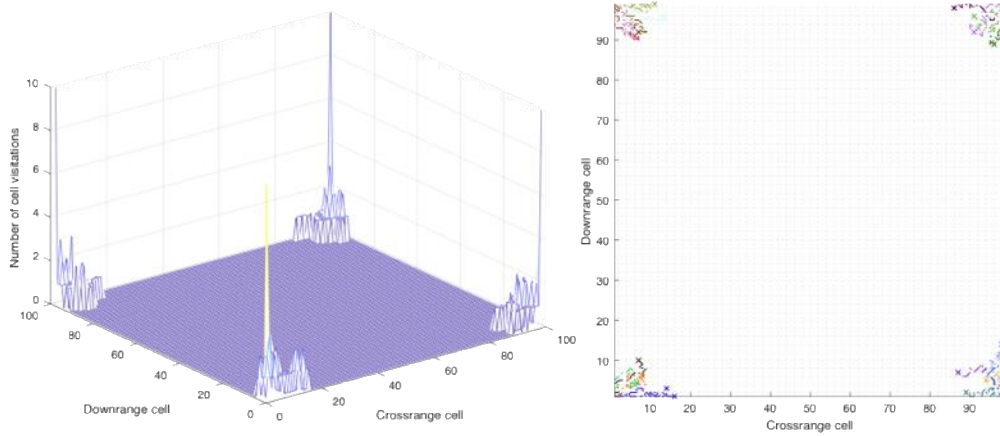


Figure 30. Snapshot of Four-Corners starting configuration.

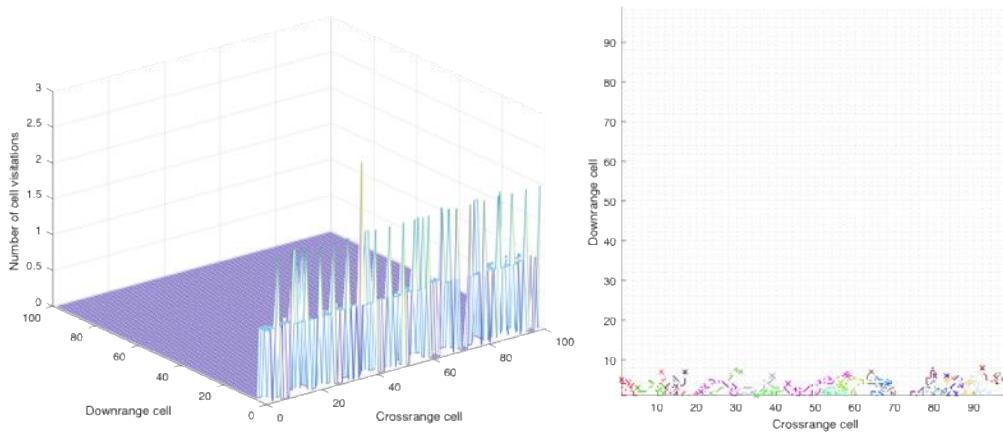


Figure 31. Snapshot of Row starting configuration.

1. Effect of Swarm Size on Starting Configurations

Figure 32 shows the effect of varying the starting configuration with 10-, 20- and 30-agent swarms. The relative effect on the lower plot is computed with respect to the single-corner entry, which happens to have the worst performance among all starting configurations.

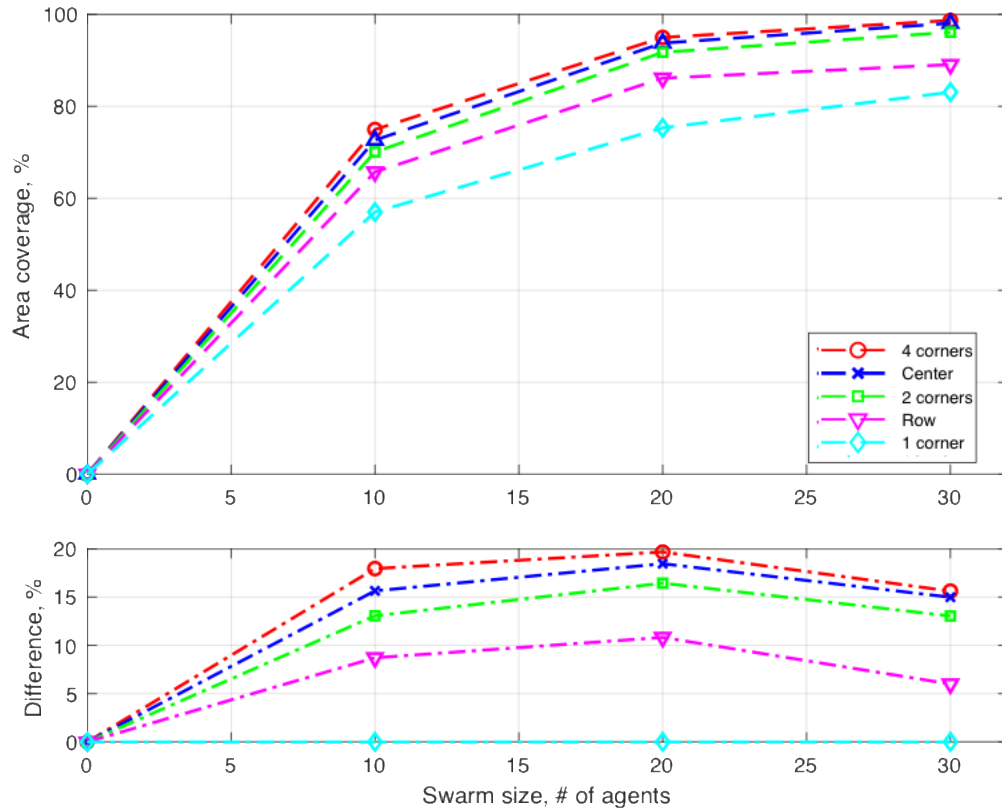


Figure 32. Coverage of various starting configurations by swarm size.

The starting configuration with all four corners as entry points turned to be the best, followed by the center and two-corner entries. Even a row-entry configuration exhibited a 5–10 percent improvement compared to a one-corner entry, which is an expected result as the agents are split into different areas thus reducing time for them to get to the unexplored area. The results also demonstrate some optimum values for swarm size, after which the positive effect seems to degrade. For this particular simulation, it was a 20-agent swarm.

2. Effect of Maximum Number of Iterations on Starting Configurations

The effect of varying the number of iterations is shown in Figure 33. Five different values—200, 400, 600, 800, and 1,000—were used in simulations with a fixed swarm size of 20 agents and the starting configurations from Figure 26.

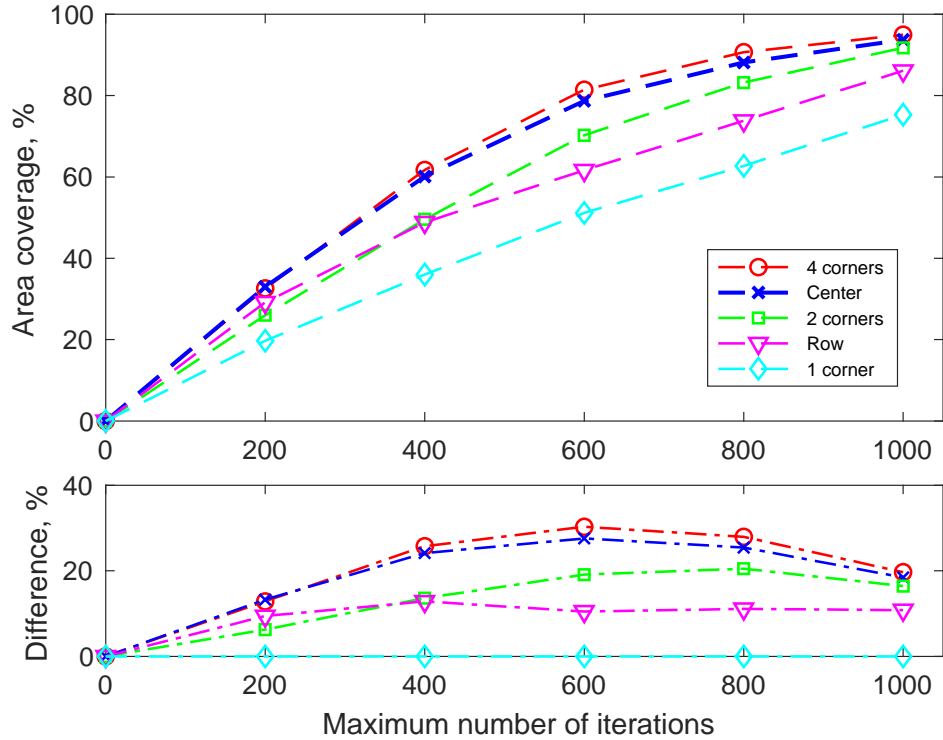


Figure 33. Coverage of various starting configurations by number of iterations (duration).

As can be seen, the simulation results are consistent with those of Figure 32. The four-corners starting configuration proved to be the best yet again. The one-corner-entry starting configuration achieved the lowest coverage regardless the number of iterations. The results beyond the 600-iteration simulation seem to yield diminishing returns for all starting configurations.

Reviewing all the findings up to this point leads one to the conclusion that a swarm consisting of 20 agents using multiple entry points into an operation area yields the best area coverage with 600 iterations. Beyond these values, saturation occurs.

E. EFFECT OF THE COLLISION AVOIDANCE CONSTRAINT

It would be interesting to see whether the inclusion of real-world constraints changes any of the aforementioned conclusions. This section shows simulation results that include collision avoidance. To assure that the collision-avoidance guidance does work, Figure 34 shows the spread of the cell distances between any two agents in a 20-agent

swarm simulation. The lower plot of Figure 34 shows the minimum distance, proving that not a single collision (distance of zero) has occurred during this simulation.

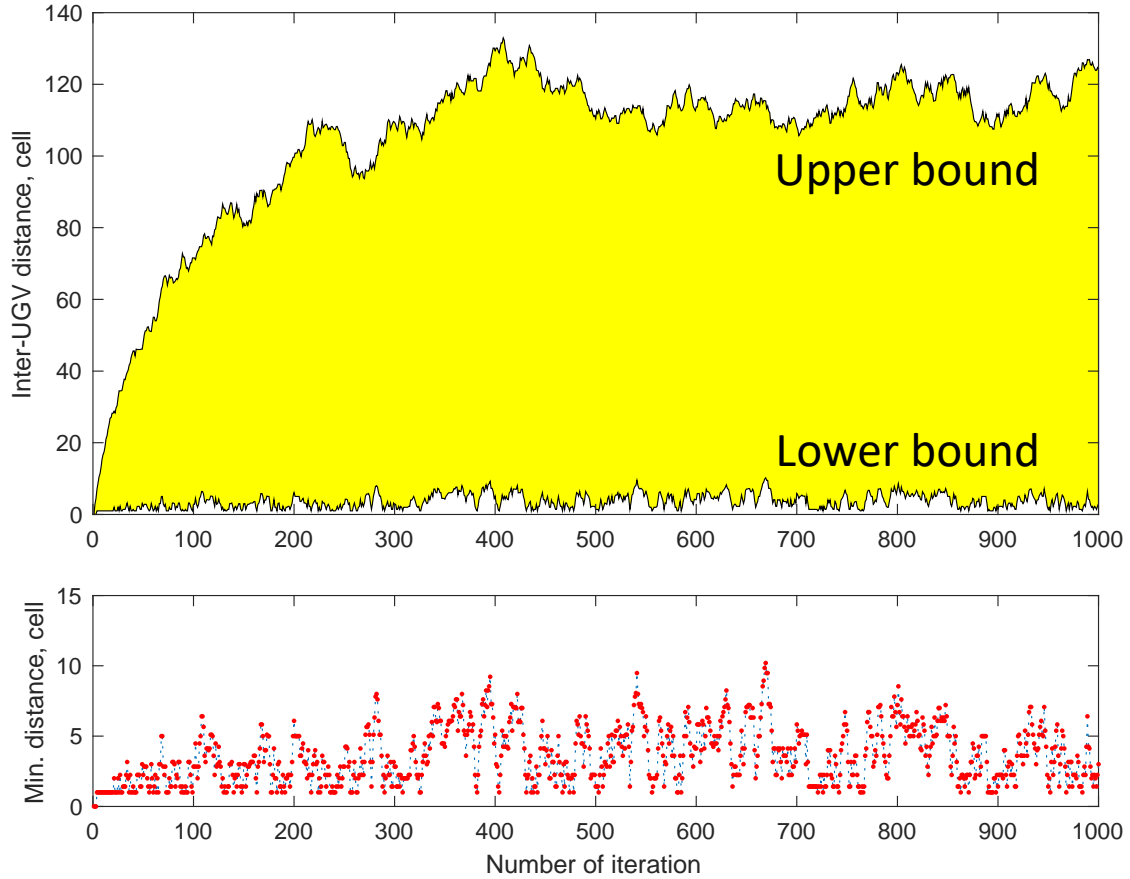


Figure 34. Maximum and minimum distances between any two agents.

The effects of various starting positions when collision avoidance is considered are presented in Figure 35. The lower plot shows some negative effects that accounting for collision avoidance produces.

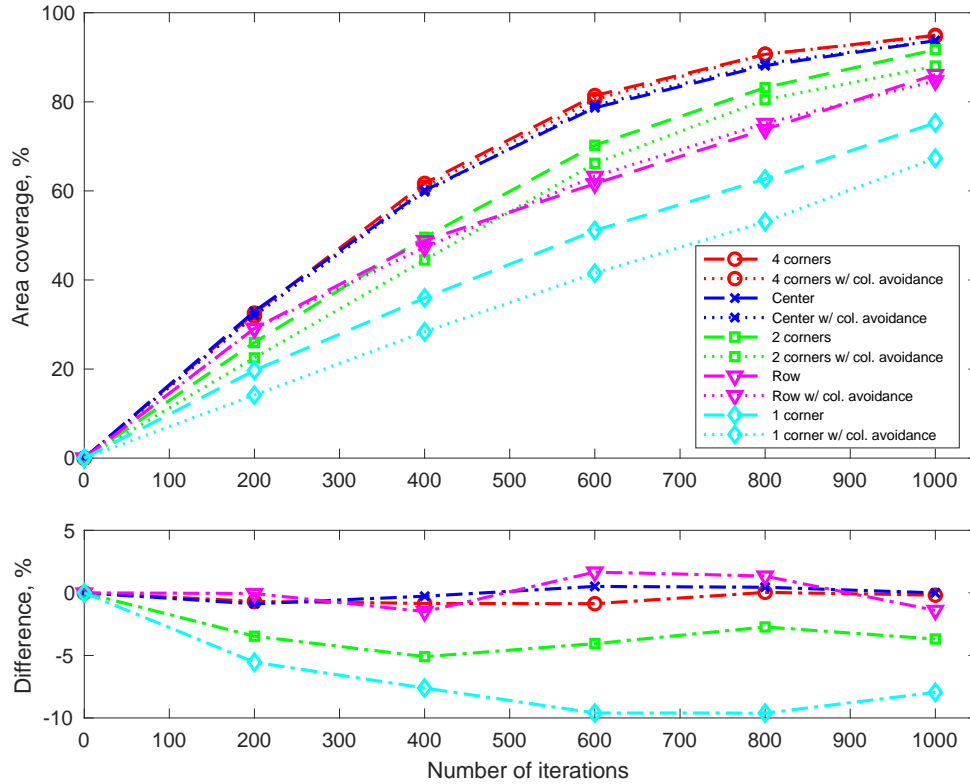


Figure 35. Effect of starting positions on coverage when incorporating collision avoidance.

Generally speaking, for most starting configurations (except single- and two-point entry types) this effect is negligible. This is likely because with the multiple starting points agents have more space to maneuver out of their initial positions. The difference in the area coverage growth rate in the beginning of the single- and two-point entry simulations suggests that the reduction is likely due to congestion that results in a queue to get out of the corner. In addition, the agents that started to move out of the corner after queuing recognize the surrounding cells as being visited, thus discouraging exploration. By comparison, the agents without collision avoidance are able to get out of the corner right from the beginning. The similar growth rate as time progresses suggests that the effect of queuing is mitigated over time after the agents spread out from their starting configuration. This conclusion is supported by Figure 36, which shows the heat map comparison of results of with collision avoidance (on the left) and without collision avoidance (on the right) for the first 50 iterations.

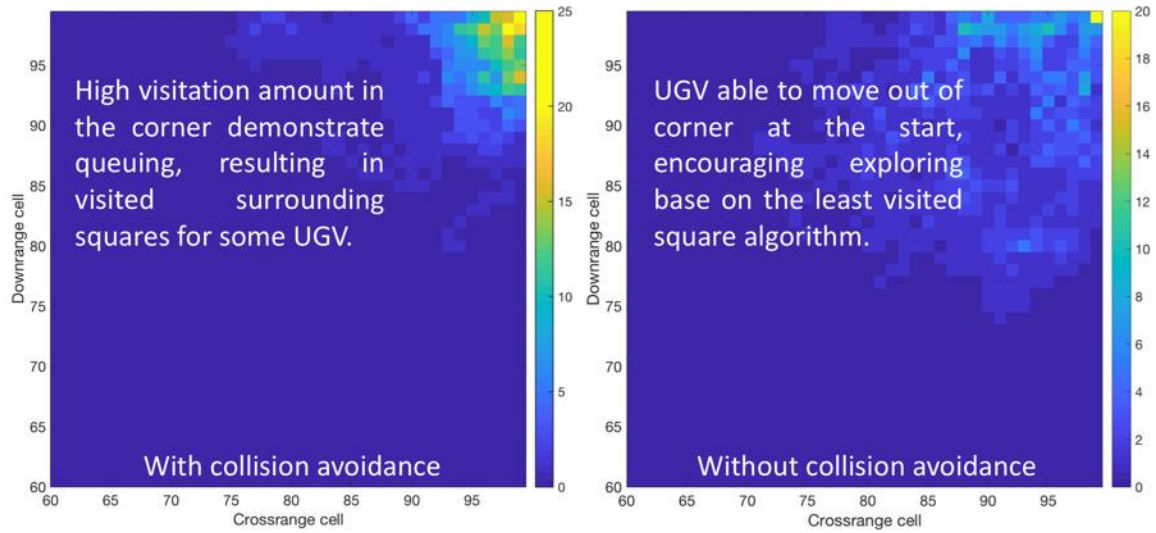


Figure 36. Heat map comparison of 50 iterations for collision avoidance (left) and without collision avoidance (right).

The queue in the four-corners starting configuration is significantly lower compared to the one-corner and two-corners configurations because fewer agents are in each corner, and therefore, these simulations are not as significantly affected by the imposition of the collision-avoidance constraint. Obviously, this conclusion might change for the larger-size swarms.

F. EFFECT OF THE NON-HOLONOMICITY CONSTRAINT

This subsection explores the effect of imposing one more real-world constraint introduced previously that has to do with agent's turn rate. Three non-holonomicity constraints of 90 degrees, 180 degrees, and 270 degrees were studied. Figure 37 to 39 show the results of UGV agents' movements under the various holonomic parameters for 100 iterations.

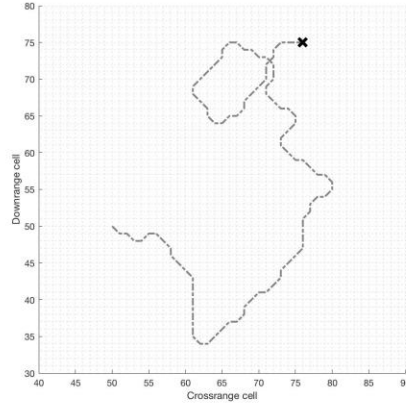
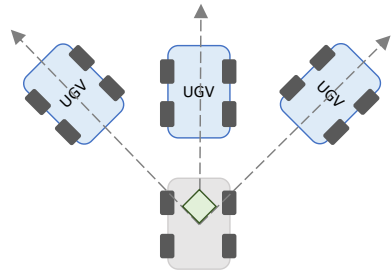


Figure 37. Non-holonomic constraint of 90 degrees.

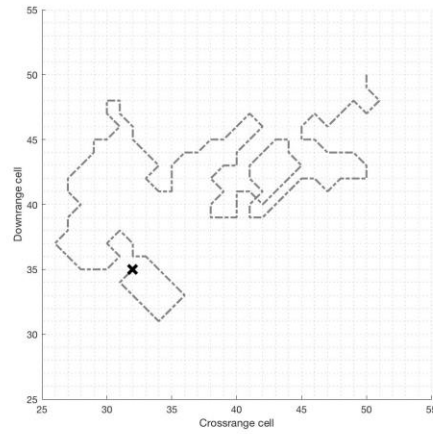
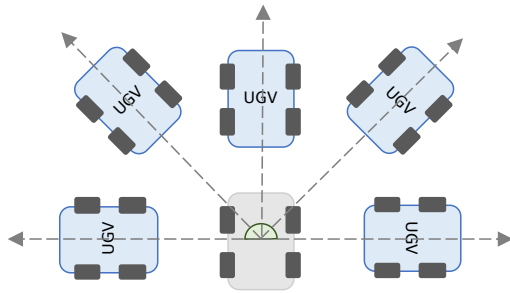


Figure 38. Non-holonomic constraint of 180 degrees.

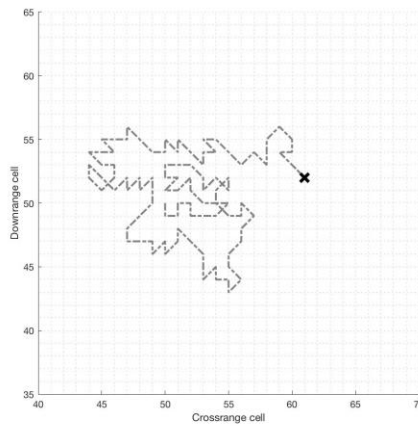
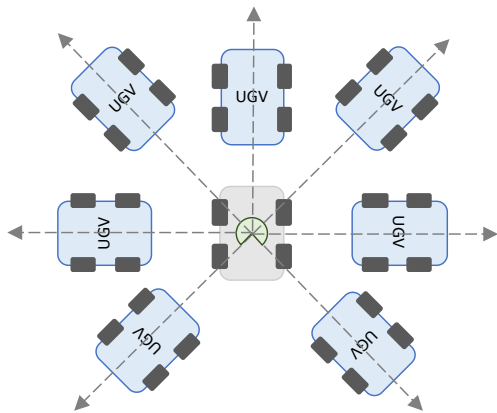


Figure 39. Non-holonomic constraint of 270 degrees.

It is observed that a non-holonomicity constraint of 90 degrees encourages exploration as it propels the UGV agent forward as compared to holonomic drive, where the agent tends to maneuver around the starting configuration.

Figures 40 to 43 show four subplots, each depicting the different non-holonomicity constraints. The first subplot in each figure shows UGV agent movement from the plan view. The second subplot shows the movement direction with reference to the map according to the following angles:

- 90 degrees (north)
- 45 degrees (north-east)
- 0 degrees (east)
- -45 degrees (south-east)
- -90 degrees (south)
- -125 degrees (south-west)
- 180 degrees (west)
- 125 degrees (north-west)

The third subplot indicates the difference in rotation angle per iteration from the agent perspective. If the agent is not turning, the rotation angle at that iteration would be zero—regardless of direction in which the agent is heading. The final subplot shows the total number of rotations made by the agent. It is generally evenly distributed since there are no obstacles and the probability of choosing an unvisited cell is random.

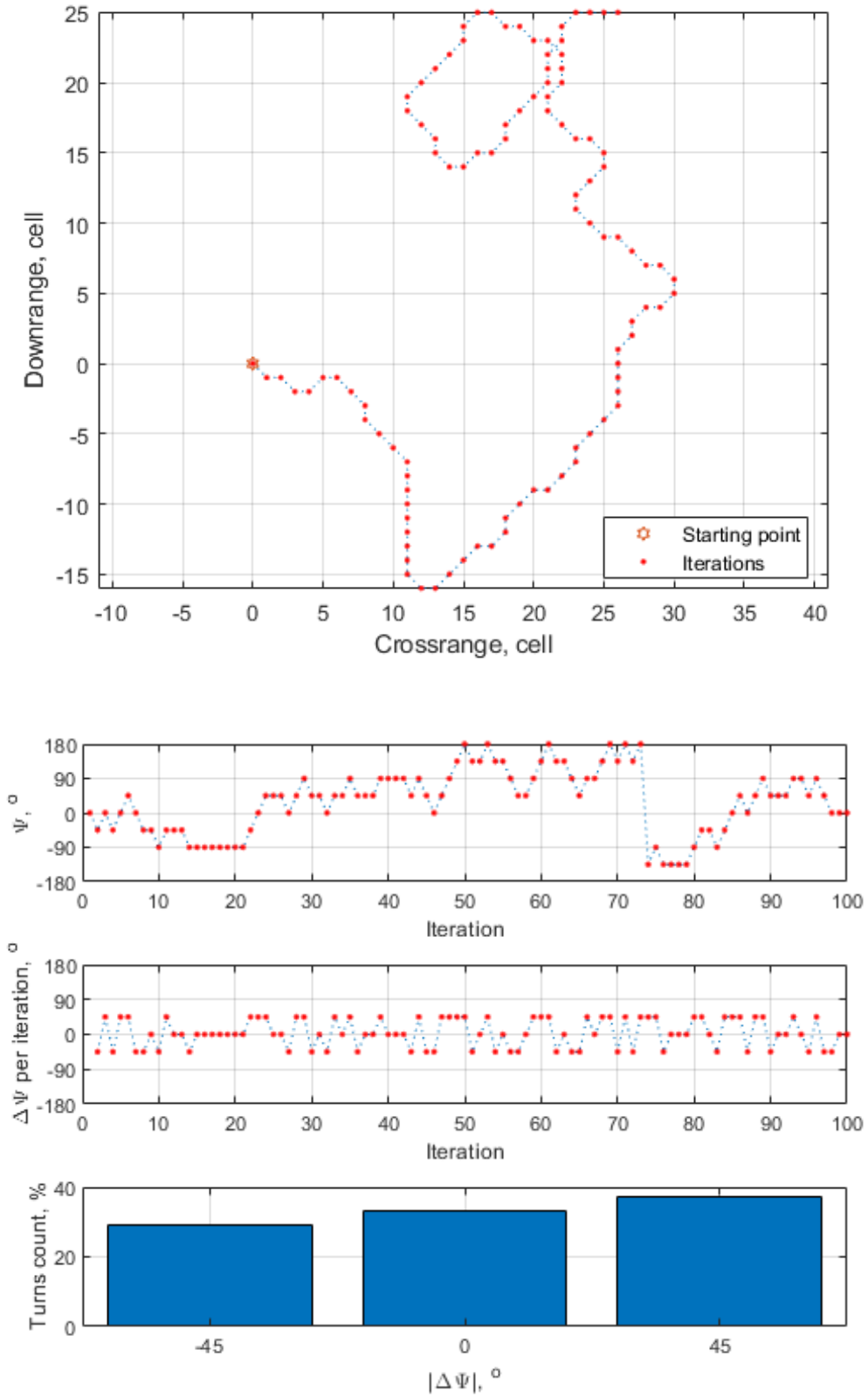


Figure 40. Non-holonomicity constraint of 90 degrees.

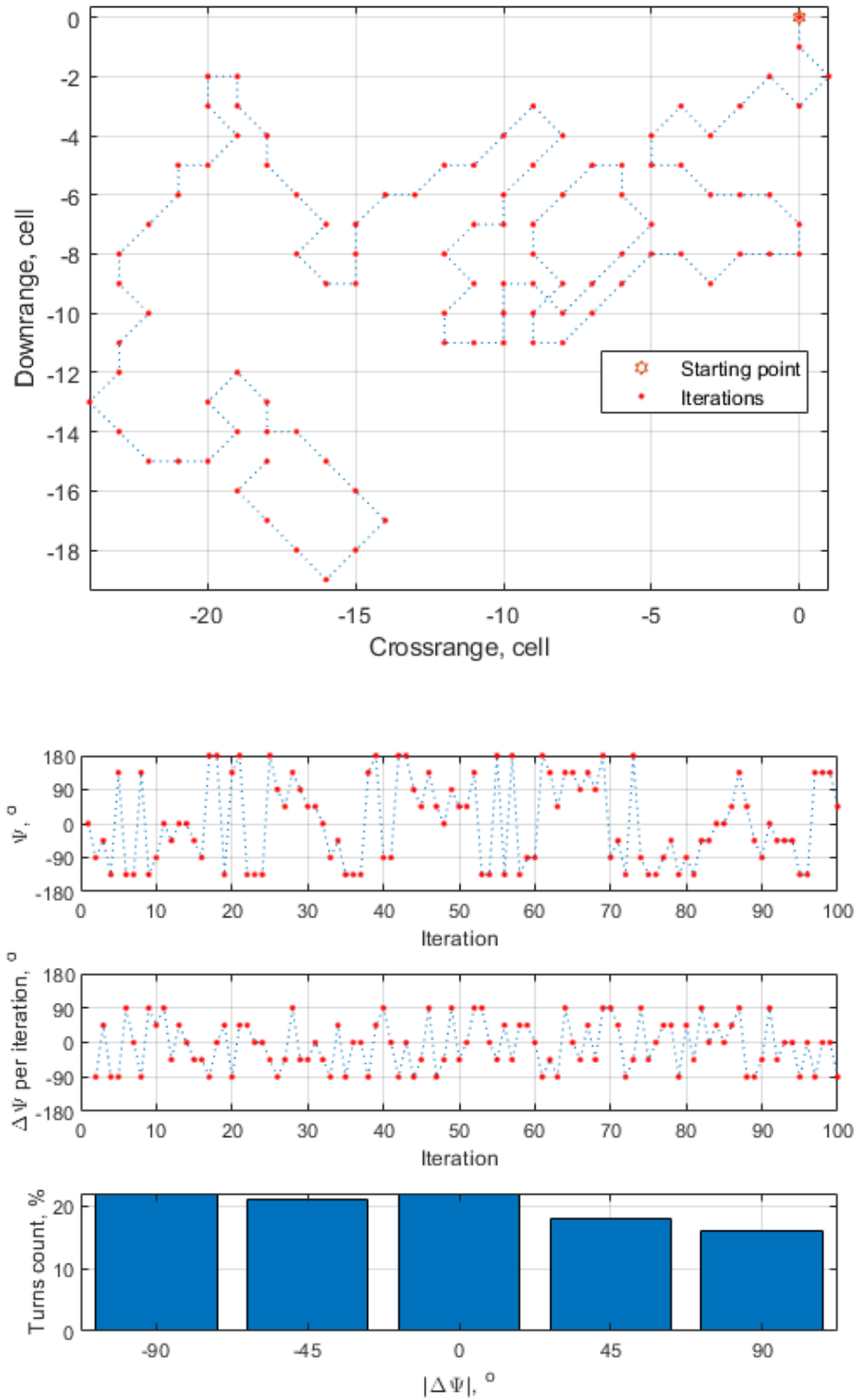


Figure 41. Non-holonomic constraint of 180 degrees.

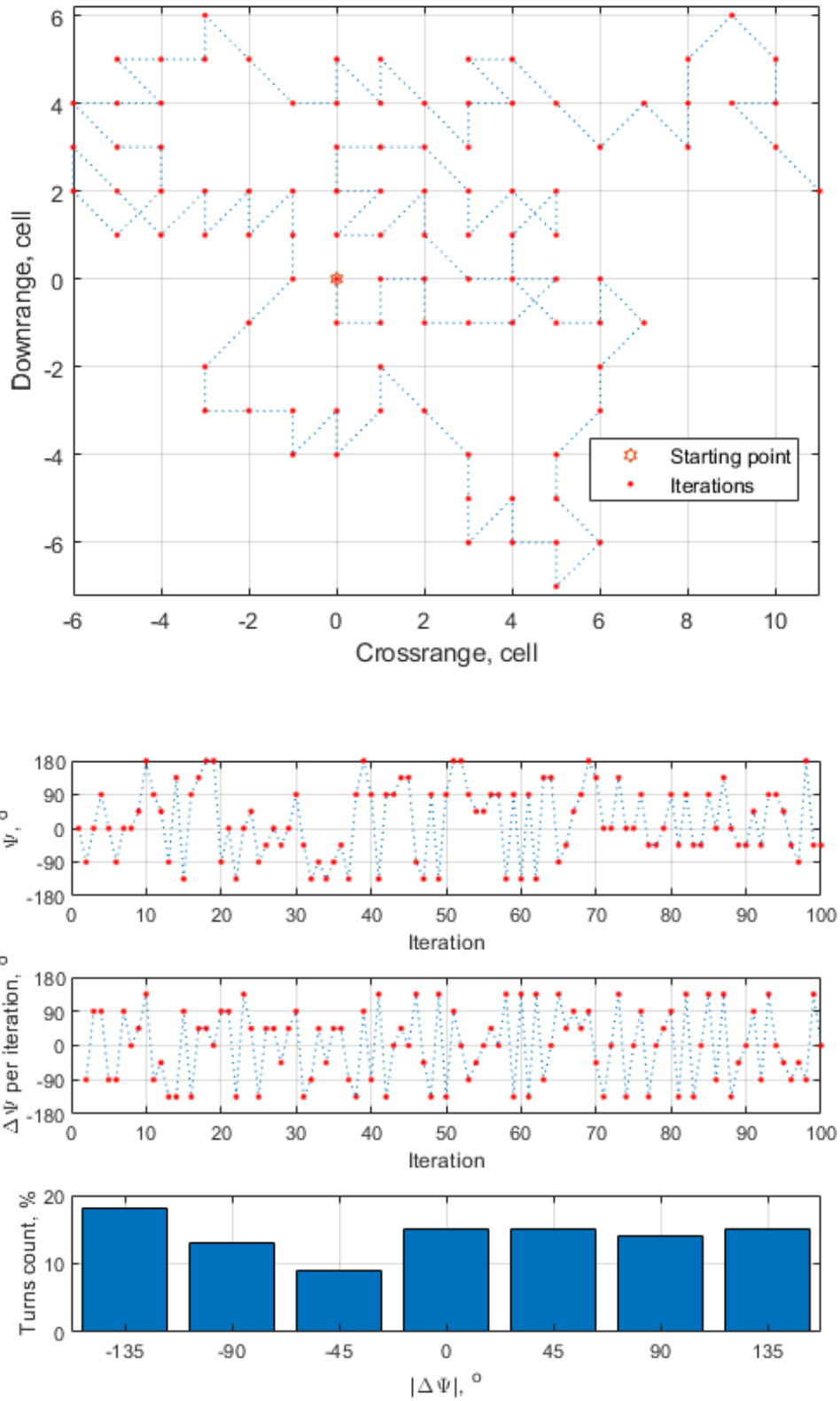


Figure 42. Non-holonomicity constraint of 270 degrees.

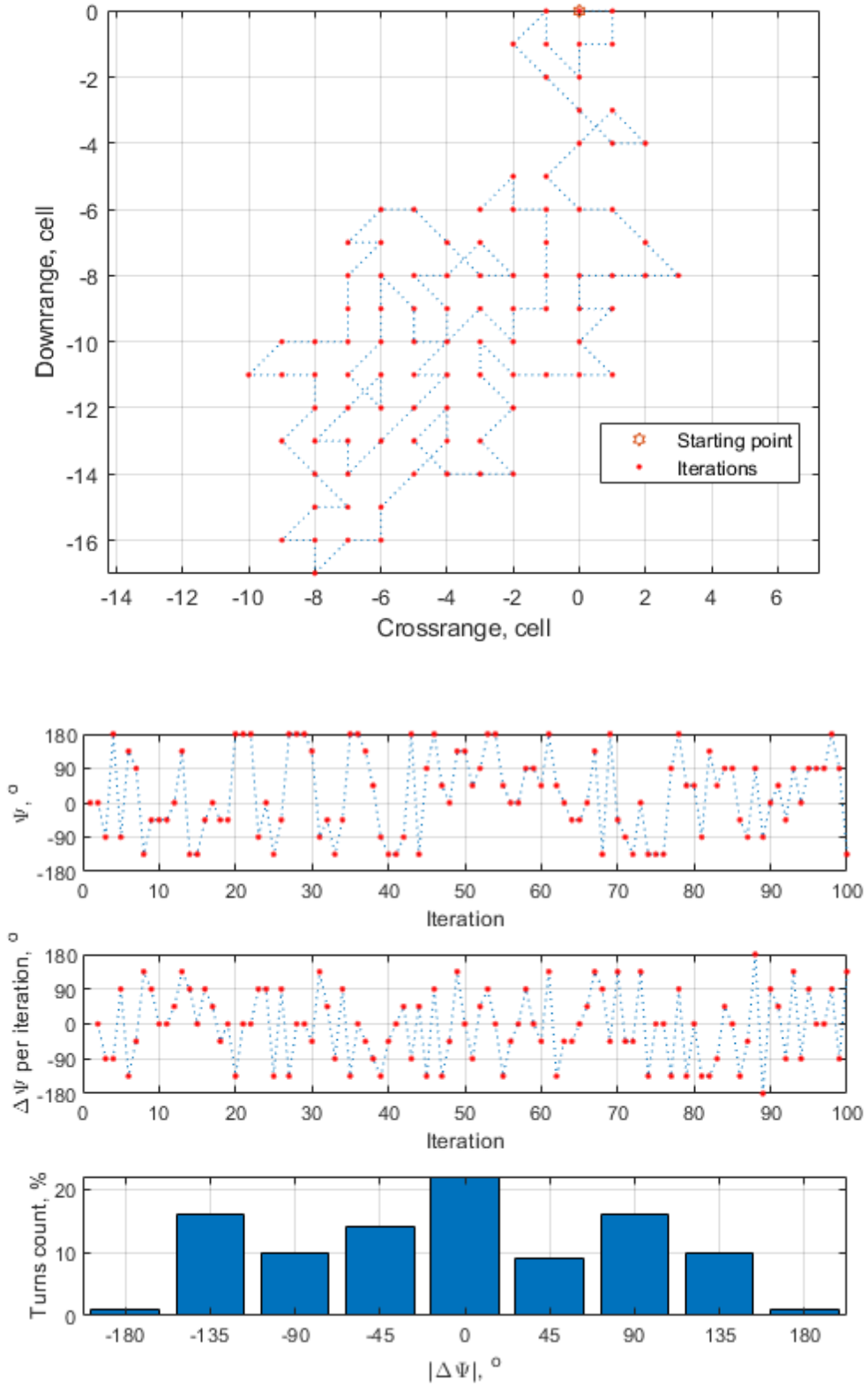


Figure 43. No non-holonomicity angle constraint.

Figure 44 shows the effect of various restriction angles of non-holonomicity constraints on coverage with a swarm size of 20 agents in the one-corner starting configuration.

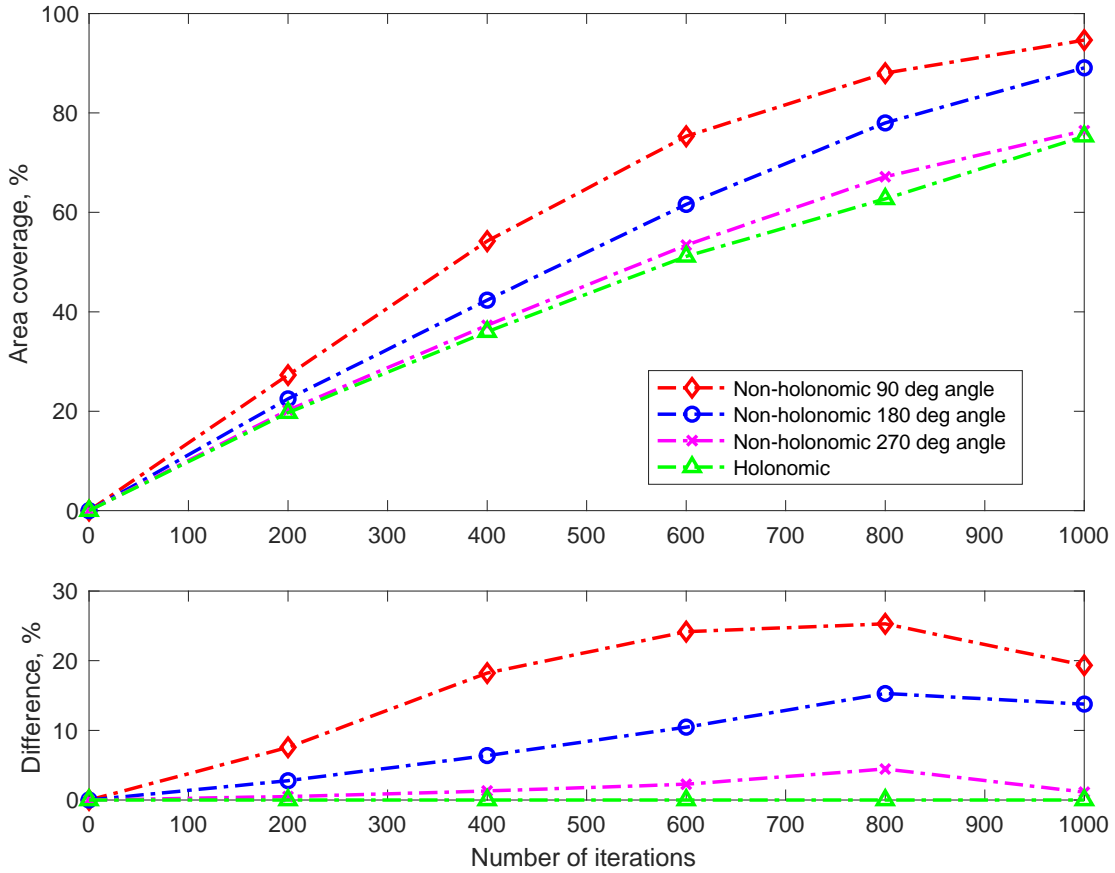


Figure 44. Effect on various non-holonomic angle constraints.

Surprisingly, area coverage increases as the non-holonomicity constraint becomes tighter. Simulation with the non-holonomicity constraint of 90 degrees provides the highest area coverage and the holonomic drive provides the lowest area coverage. This finding suggests that this constraint actually encourages exploration, and thus leads to fuller coverage over time.

The effect of non-holonomic drive on various starting positions is also investigated. Figure 45 shows that the non-holonomicity constraint of 180 degrees improves area coverage on all starting configurations.

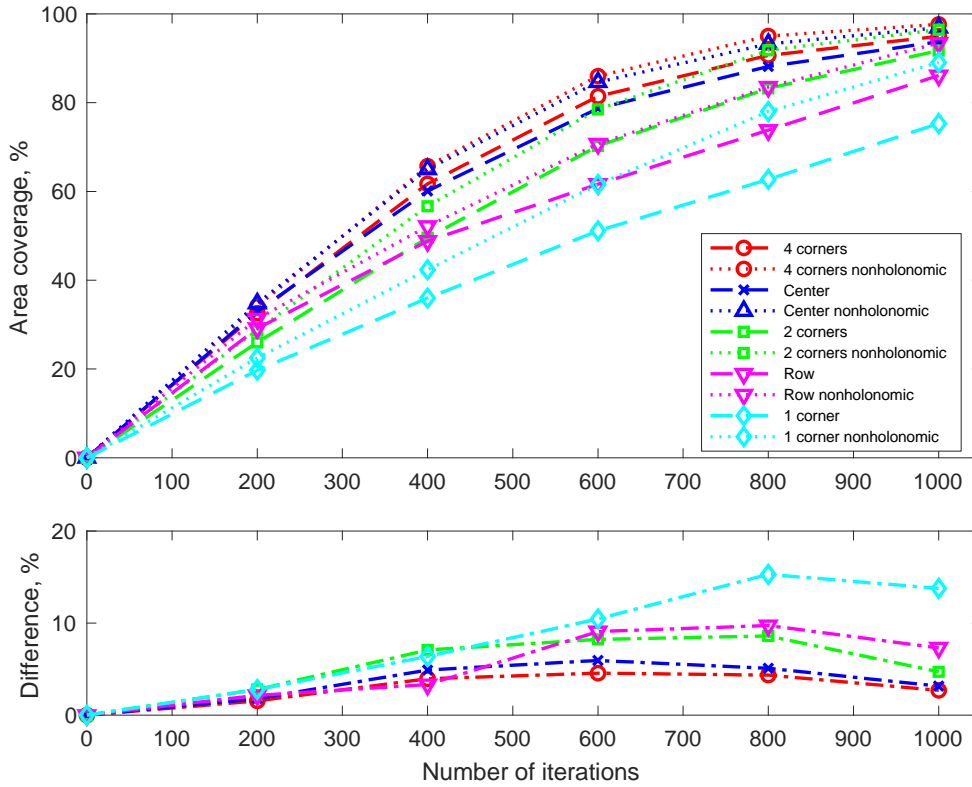


Figure 45. Effect of a non-holonomicity constraint of 180 degrees and various starting configurations.

We can conclude in this section that for the purpose of exploration, non-holonomic drive behaviors should not be seen as a limitation but a tool to encourage either global exploration or local area search. Holonomic drive behaviors is subsequently investigated in an urban area as well as an indoor simulation in the next sections.

G. URBAN OUTDOOR SEARCH OPERATIONS

Now that all aspects of the developed algorithm have been studied in the open-space environment, the LVC guidance needs to be evaluated for a more realistic environment like the ones shown in Figure 16 and 17.

1. Effect of Various Starting Configurations

These simulations involve a 20-agent swarm obeying collision-free operations. Three possible starting configurations for the outdoor environment are depicted in Figure 46. Configuration 1 (left) represents a scenario where all agents are deployed from the same location, while Configuration 3 (right) utilizes three entry points.

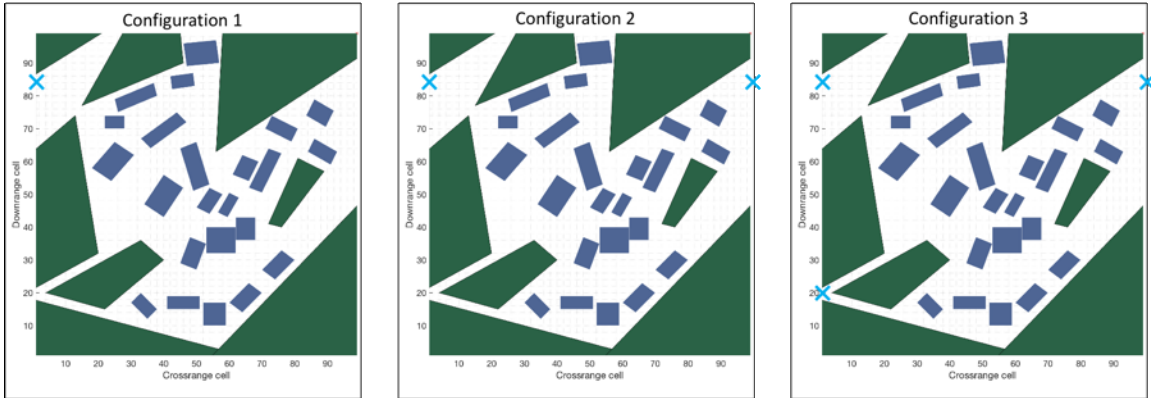


Figure 46. Starting configurations of UGV agents for urban outdoor operations.

The result shown in Figure 47 supports the previous findings that multiple launch sites encourage faster and, therefore, better area coverage compared to a single entry. For 1,000 iterations, there is a 40 percent improvement for Configuration 3 (multiple launch sites) compared to Configuration 1 (single launch site). The improvement for four-corners and one-corner starting configurations in an open space environment is approximately 20 percent. This finding seems to suggest that the benefit of multiple launch sites is amplified when obstacles are present in the environment.

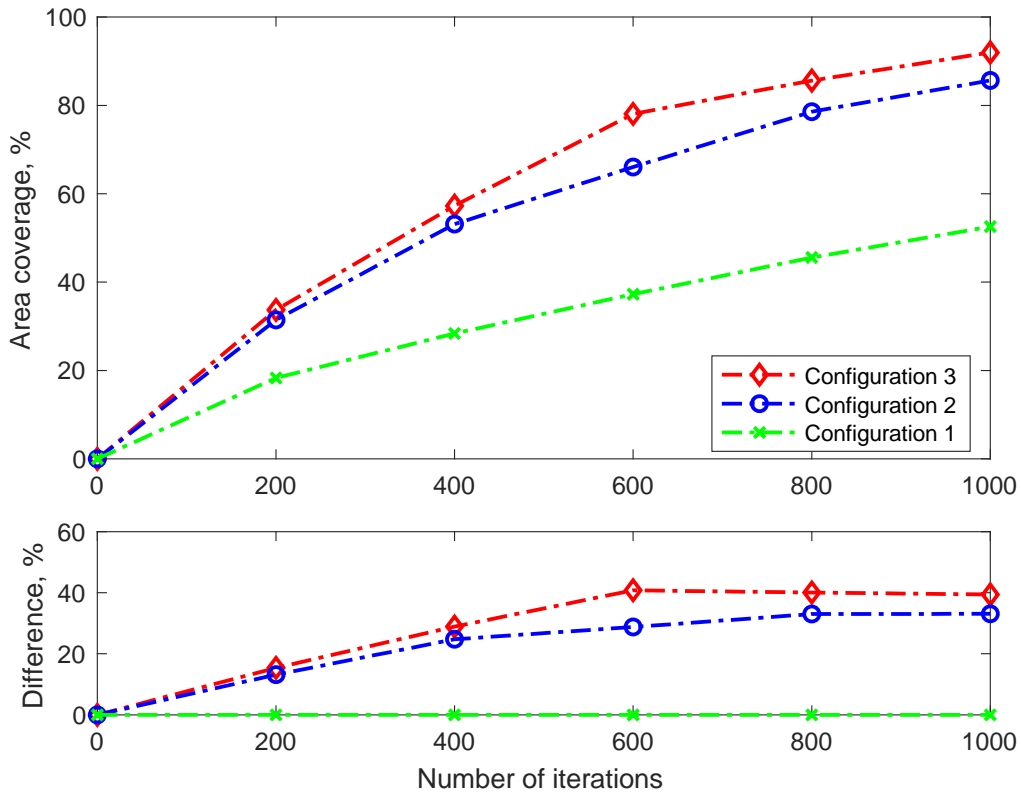


Figure 47. Effect of starting configuration on area coverage for urban operation.

2. Effect of the Non-Holonomicity Constraint

The effects of non-holonomicity constraint were also studied for a single launch site. The effect of imposing the non-holonomicity constraint in this case, presented in Figure 48, is similar to that of the open-space case of Figure 44.

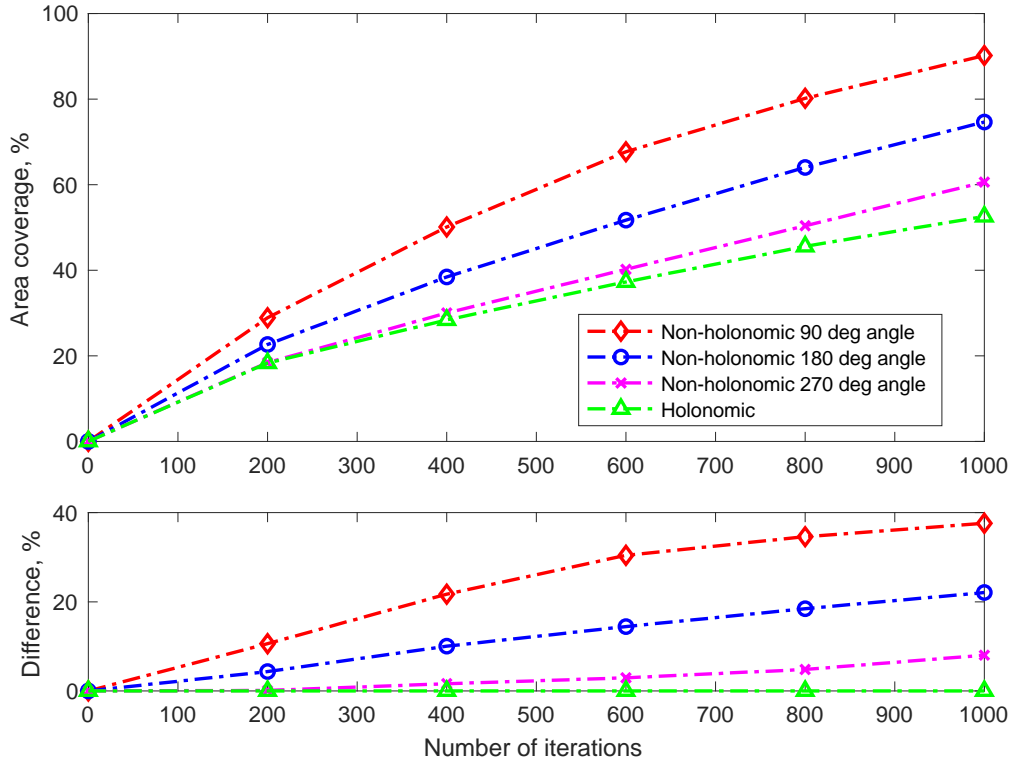


Figure 48. Effect of non-holonomic angle restriction on area coverage for urban scenario.

The effect of non-holonomicity constraint is clearly seen in Figure 49, which shows the bird's-eye-view trajectories of all 20 agents being holonomic (on the left) and non-holonomic (on the right). Yet again, imposing the non-holonomicity constraint leads to the fuller area coverage across all iterations.

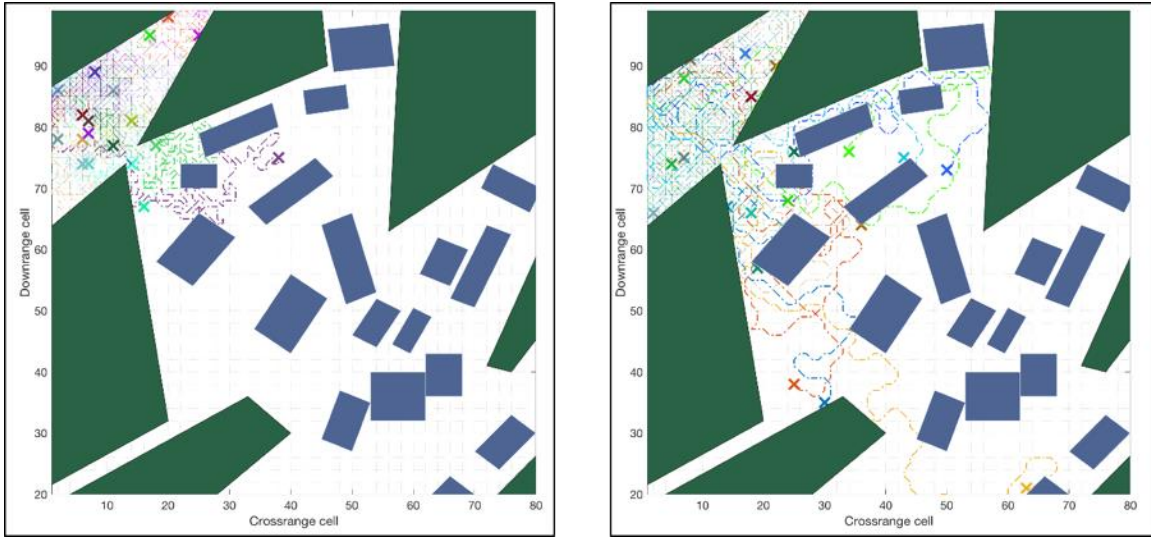


Figure 49. Holonomic (left) versus non-holonomic drive with a 90-degree constraint (right).

Compared to the results of the open-space simulation depicted in Figure 44, the positive effect is more pronounced. The holonomic Blue agents (left of Figure 49) tend to stay within a confined area as they are further restricted by obstacles. This seems to suggest that the benefit of tightening the non-holonomicity constraint is amplified when obstacles are present. Table 3 compares the improvement from holonomic to non-holonomic drive for both open space and urban operations.

Table 3. Comparison of the improvement (area coverage) in open space and outdoor urban environments with the effect of non-holonomic constraint.

Iteration	Improvement (% of area coverage) from no constraint to 270° non-holonomic constraint		Improvement (% of area coverage) from no constraint to 180° non-holonomic constraint		Improvement (% of area coverage) from no constraint to 90° non-holonomic constraint	
	Open Space	Urban outdoor	Open Space	Urban outdoor	Open Space	Urban outdoor
200	0.50	0.10	2.78	4.33	7.57	10.59
400	1.30	1.62	6.38	10.03	18.23	21.71
600	2.27	2.93	10.45	14.45	24.17	30.42
800	4.46	4.82	15.29	18.46	25.30	34.60
1000	1.13	8.03	13.76	22.07	19.34	37.62

H. INDOOR SEARCH OPERATIONS

The effect of non-holonomic drive for indoor operations is discussed in this section. The assumption for indoor the environment would be that there is only one entrance into the room and thus there would only be one starting configuration. Figure 50 shows the entrance to the room, which would be the starting configuration for the UGV agents.

Similar to the urban environment, analysis was done using a fixed swarm size of 20 agents with collision avoidance.

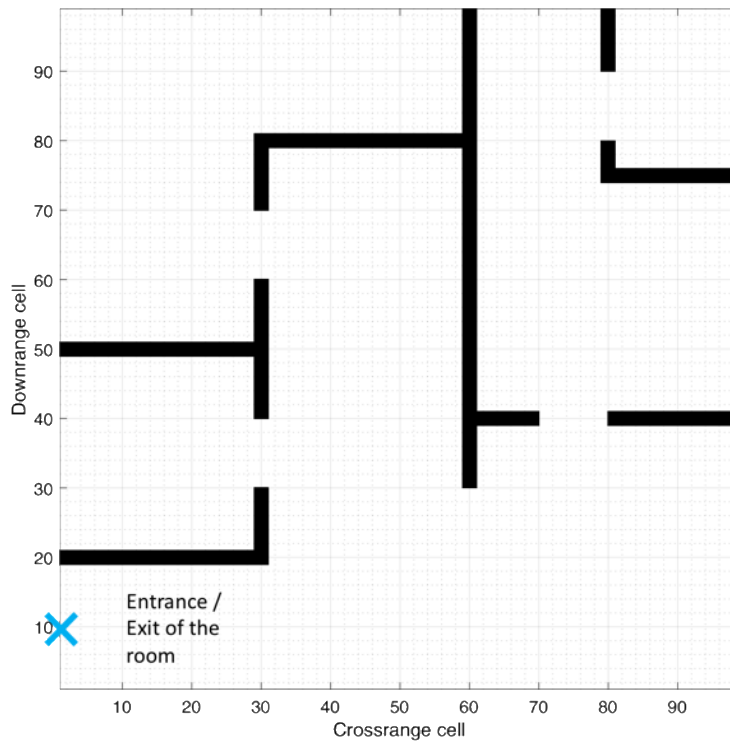


Figure 50. Starting position (entrance) to the indoor environment.

Figure 51 shows the results of the effect of non-holonomic drive for indoor operations, and Table 4 compares the improvement from holonomic drive to non-holonomic drive for open space and indoor operations.

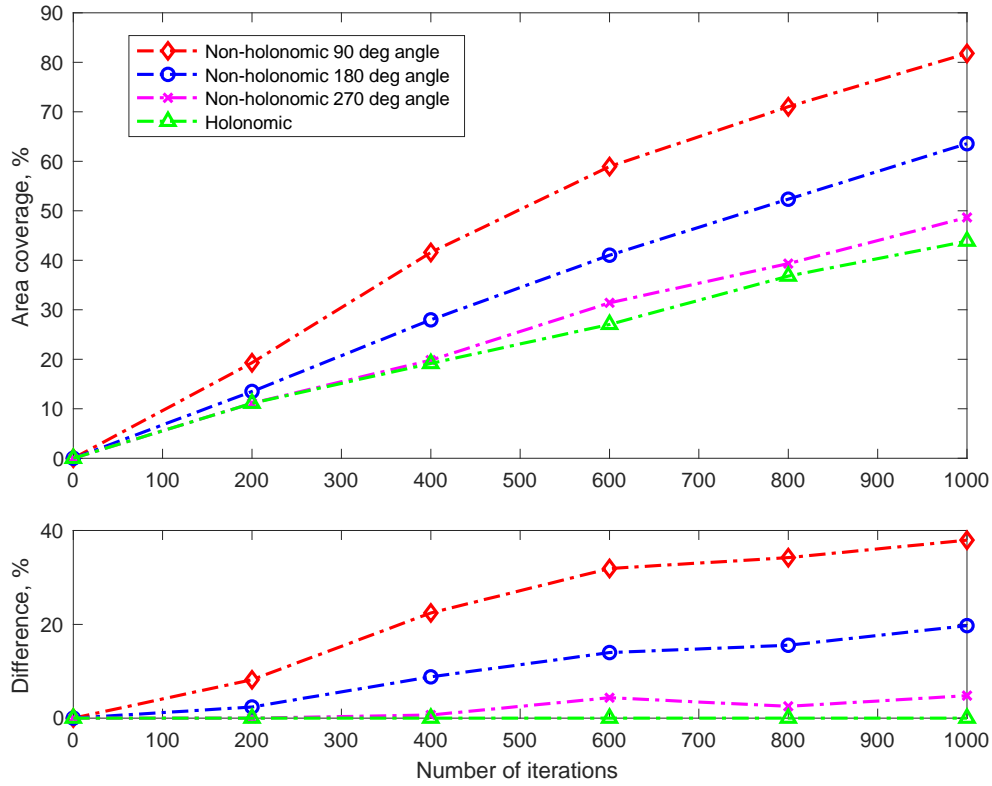


Figure 51. Effect of non-holonomic drive on area coverage for indoor operations.

Table 4. Comparison of the improvement (area coverage) in open space and indoor environments with effects of non-holonomic constraint.

Iteration	Improvement (% of area coverage) from no constraint to non-holonomic drive (270°)		Improvement (% of area coverage) from no constraint to non-holonomic drive (180°)		Improvement (% of area coverage) from no constraint to non-holonomic drive (90°)	
	Open Space	Indoor	Open Space	Indoor	Open Space	Indoor
200	0.50	0.00	2.78	2.38	7.57	8.19

	Improvement (% of area coverage) from no constraint to non-holonomic drive (270°)		Improvement (% of area coverage) from no constraint to non-holonomic drive (180°)		Improvement (% of area coverage) from no constraint to non-holonomic drive (90°)	
400	1.30	0.67	6.38	8.81	18.23	22.43
600	2.27	4.36	10.45	13.98	24.17	31.90
800	4.46	2.52	15.29	15.54	25.30	34.21
1000	1.13	4.79	13.76	19.71	19.34	37.94

As seen from the results shown in Figure 50 and Table 4, a similar conclusion can be drawn in the case of indoor search operations as well.

The results from the previous two sections reveal that the non-holonomic drive is able to produce even better results when obstacles are introduced. The UGV agents with holonomic drive tend to stay within a confined area as they are further restricted by obstacles.

I. EFFECTIVENESS OF ALVC GUIDANCE

This section investigates the effect on area coverage when the ALVC algorithm is used. Figure 52 shows a snapshot at the 700th iteration, with the trail appearing from 550th iteration onwards. The behavior of the UGV agents heading towards the remaining few unvisited cells in the grid can be observed.

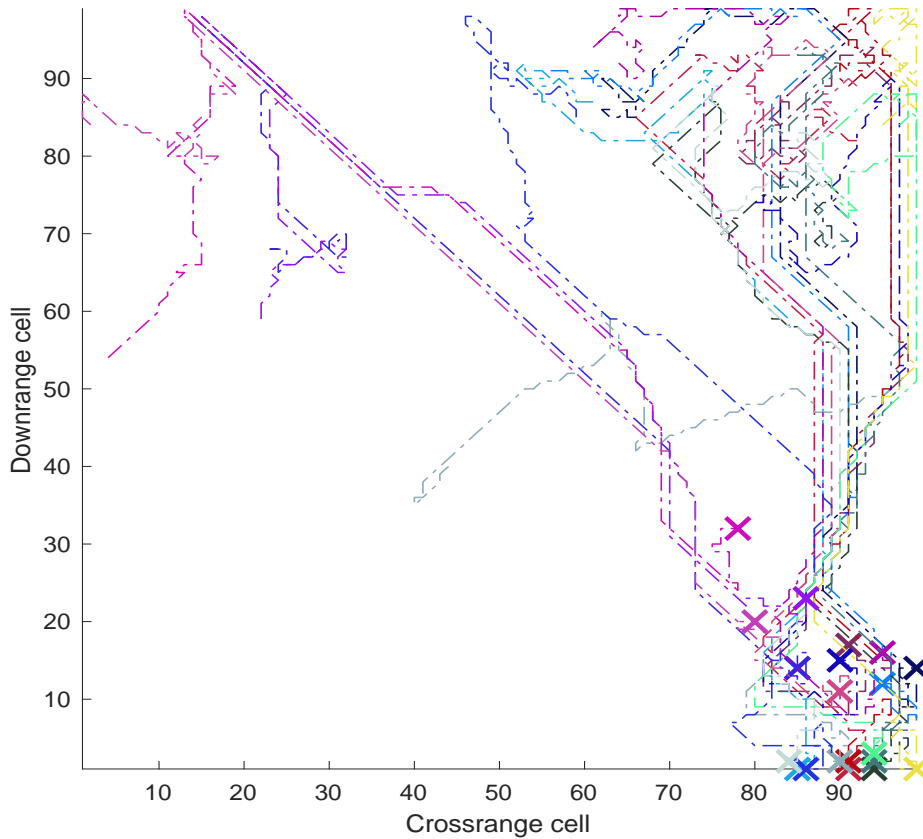


Figure 52. Snapshot of the last 100 iterations of a simulated run using ALVC guidance.

1. Holonomic Drive

During the study of this section, a limitation of the ALVC algorithm was identified. Imposing a non-holonomic drive restriction of 90 degrees would cause the algorithm to enter an infinite circle loop—in some cases, around an unvisited square—as shown in Figure 53. The LVC algorithm would not enter such a state because the UGV agents access only to the allowed cells. By contrast, the ALVC algorithm grants UGV agents access to all surrounding cells and targets the closest one. Thus, only the effect of non-holonomicity constraints of 180 degrees and 270 degrees were explored.

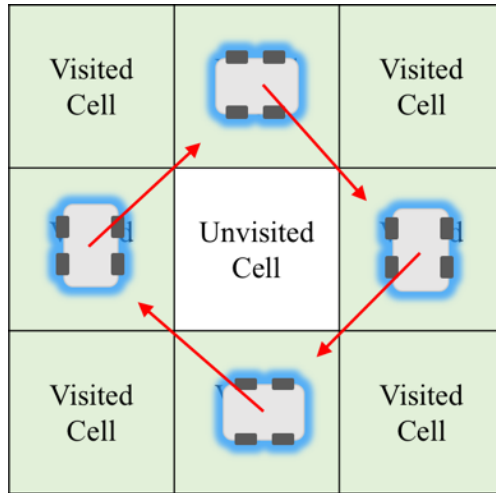


Figure 53. Infinite circle loop around an unvisited square.

The effects of non-holonomicity constraints of 180 degrees and 270 degrees on area coverage are shown in Figure 54, where subplot 2 shows the difference in area coverage from holonomic drive.

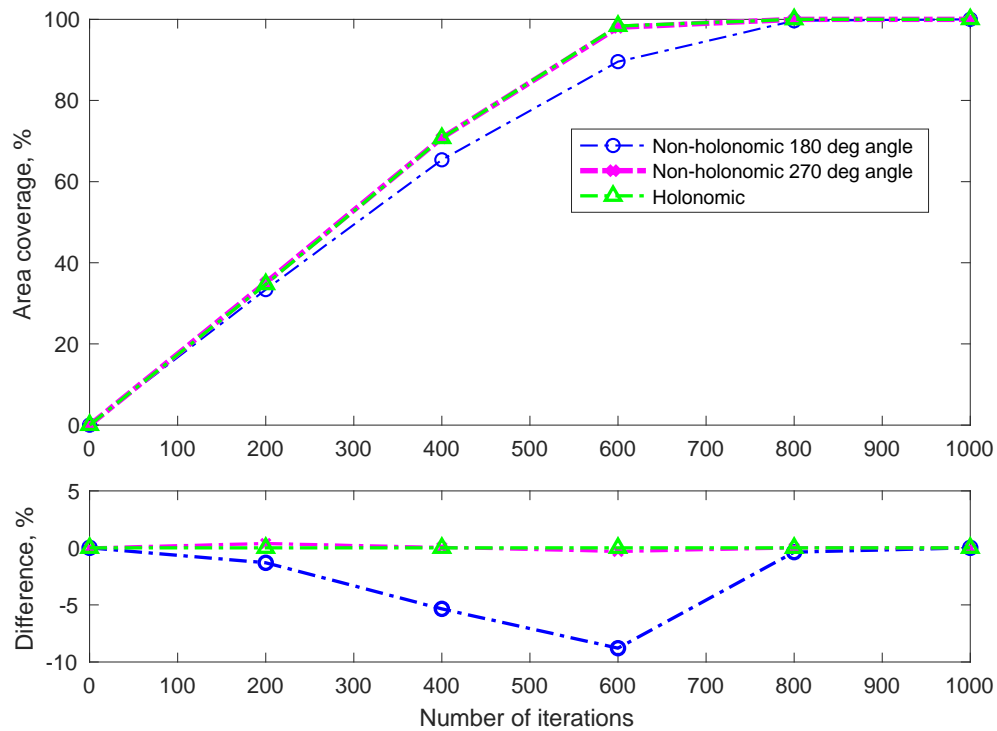


Figure 54. Effect of non-holonomic drive on the ALVC algorithm.

The data suggest that the introduction of non-holonomic drive decreases area coverage slightly unlike previous findings for the LVC algorithm. This is likely due to the fact that the non-holonomicity constraints imposed restricts the freedom to move directly to the targeted cell in every iteration for the ALVC.

2. Environment

As was done to evaluate the LVC guidance algorithm, three different environments—open space, outdoor, and indoor urban environments—were simulated for the ALVC guidance algorithm, and the results were compared against those for the LVC algorithm, as shown in Figure 55.

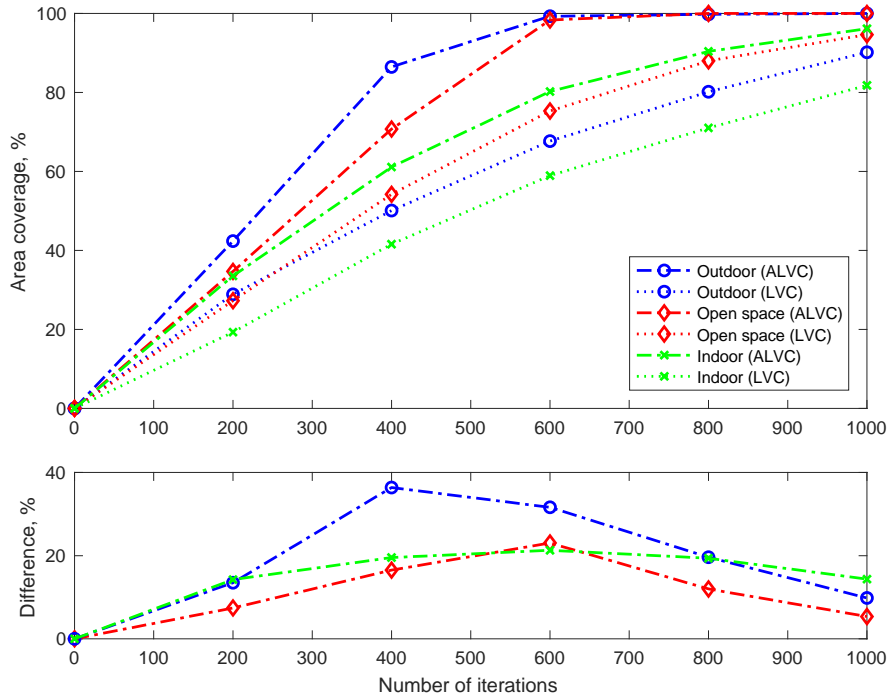


Figure 55. Comparison between results of LVC and ALVC algorithms for three environments.

The results suggest that despite the limitation of the non-holonomicity constraint of 90 degrees, the ALVC algorithm is able achieve greater area coverage in all three environments. Further analysis on the ALVC guidance is discussed in the next chapter.

V. STUDY OF THE TRACK AND ENGAGE PHASE

The previous chapter dealt with an area coverage as a single swarm objective. Now, we extend the aforementioned simulations to include the Red forces, thus adding one more objective of engaging an opponent. Because of this, once any Red agent is detected the Blue agents largely abandon the primary objective and pursue the second one. As a result of engagements, the swarm size becomes variable (decreasing).

All simulations presented in this section were conducted for a 20-agent Blue swarm acting against five Red agents. Figure 56 shows the initial setup for the open-space, outdoor, and indoor urban environments. Blue agents had a 90-degree non-holonomic drive restriction and obeyed the collision-free constraint during the search phase.

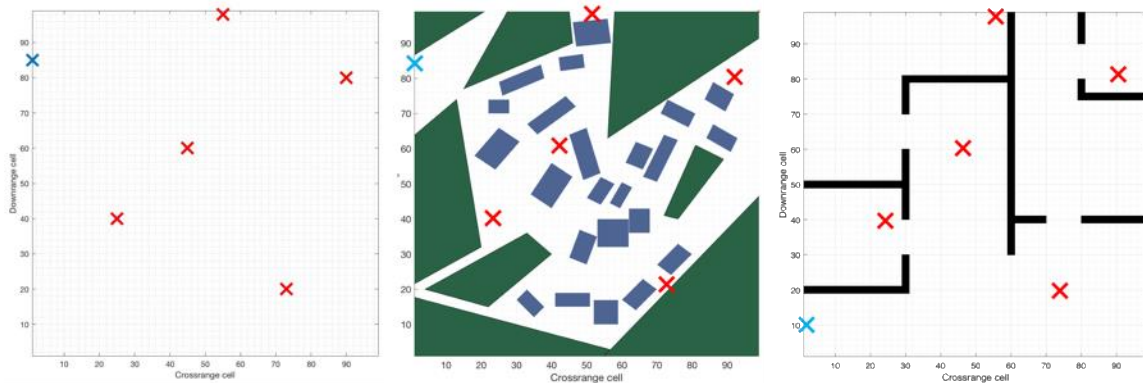


Figure 56. Starting configurations of Blue and Red forces for open space (left), outdoor (center), and indoor (right) urban operations.

The two measures of effectiveness in this case were the number of iterations required to kill all agents of the either side as well as the number of agents that survived the engagement.

The following parameters are varied and their effects on the two measures of effectiveness are investigated.

- Addition of PSO guidance

- Varying Blue forces detection range
- Varying holonomicity constraint (tracking) for Blue forces
- Varying probability of kill (P_k) for both forces
- Varying kill distance
- Changing kill sequence
- Introduction of outdoor and indoor urban environment
- Effectiveness of the Advanced Least Visited Cell guidance

A. EFFECTIVENESS OF ADDED PSO GUIDANCE

In this section, the effect of the addition of the PSO guidance component is investigated against the original LVC algorithm during the track and engage phase. PSO guidance would be triggered for Blue forces when a Red agent is within any Blue agent's detection range. The Blue forces will evaluate their current positions with the detected enemy and compute its velocity vector accordingly to swarm towards the detected enemy.

As an illustrative example Figure 57 presents snapshots of open-space simulations during the track and engage phase featuring trajectories of 10 agents in the case PSO guidance is not activated (on the left) and is activated (on the right). In the first case, Blue agents simply wander around providing fuller area coverage but not necessarily staying engaged with Red agents when they are detected; in the second case, three Red agents have been engaged within the same number of iterations.

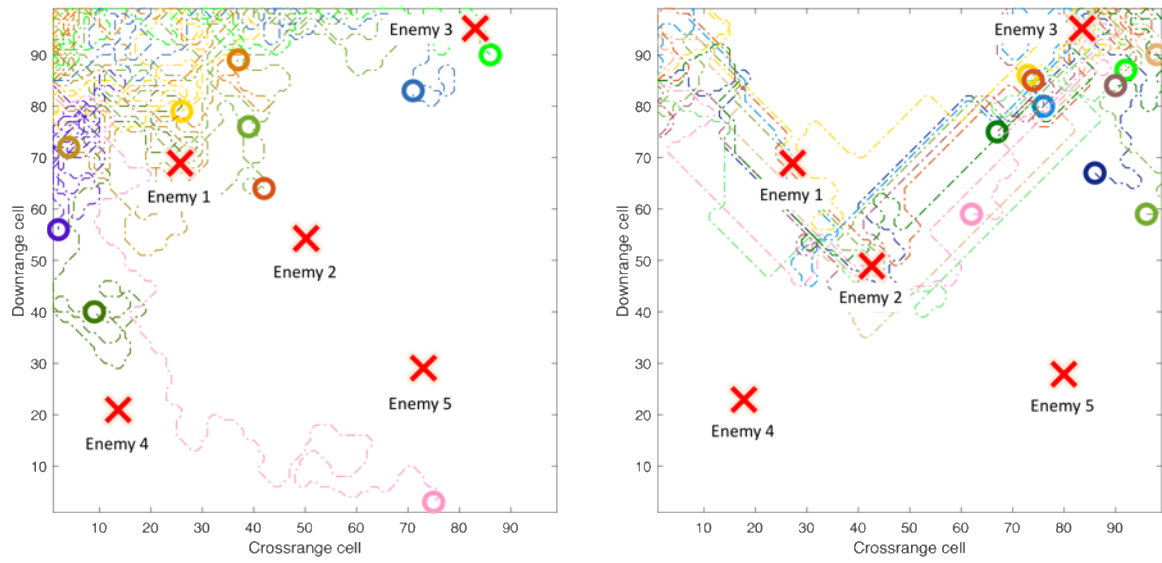


Figure 57. Trajectory comparison between LVC (left) and PSO (right) guidance during the track and engage phase.

Running this simulation 30 times produces a statistically-verified estimate of the PSO algorithm addition effectiveness. To this end, the box plot of Figure 58 shows the number of iterations needed to end the engagement (when all agents of either side are destroyed), and Figure 59 compares the number of casualties sustained at the end of the battle. The green crosses in the box plot show the mean values while the red lines denote the median values.

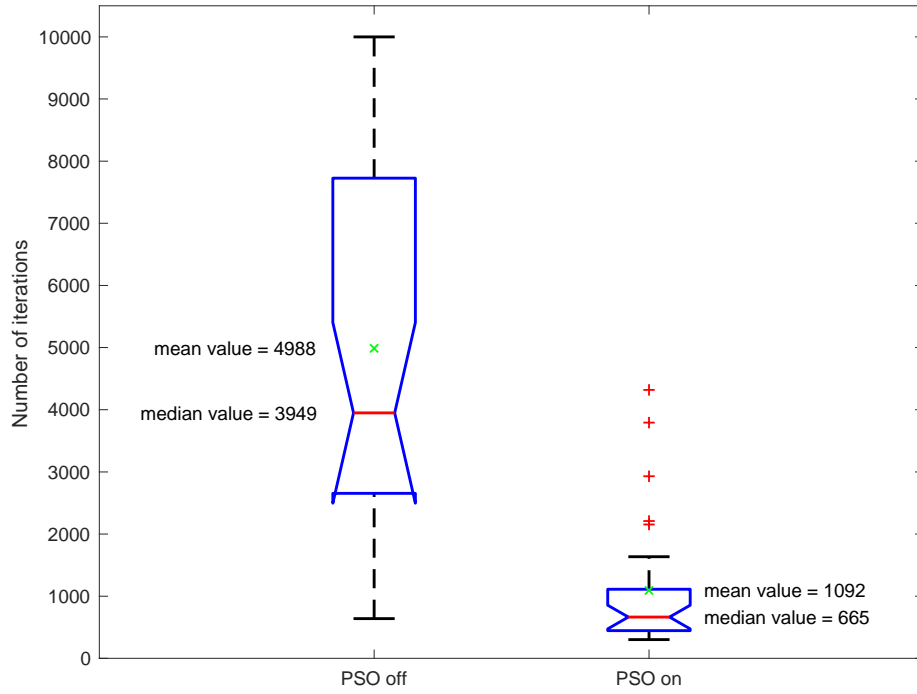


Figure 58. Number of iterations needed for a battle with and without PSO guidance.

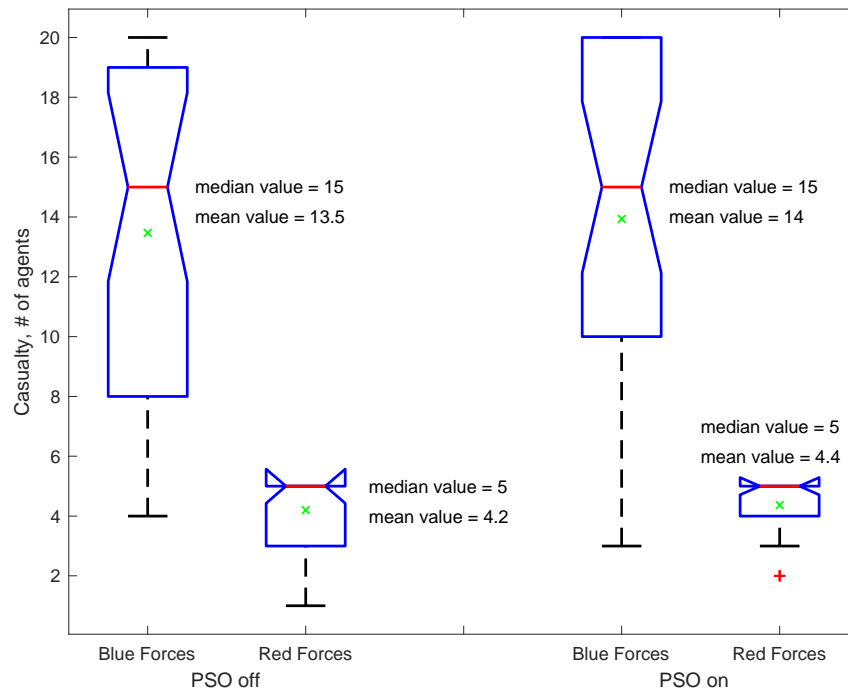


Figure 59. Number of casualties with and without PSO guidance.

Figure 60 and 61 show the pairwise comparison done by analysis of variance (ANOVA)—specifically, the Fisher’s least significant difference (LSD) test—on the number of iterations needed for a battle with and without PSO guidance.

ANOVA Table					
Source	SS	df	MS	F	Prob>F
Groups	2.27745e+08	1	2.27745e+08	40.88	3.02881e-08
Error	3.23096e+08	58	5.57063e+06		
Total	5.50841e+08	59			

Figure 60. ANOVA table for the number of iterations needed for a battle with and without PSO guidance.

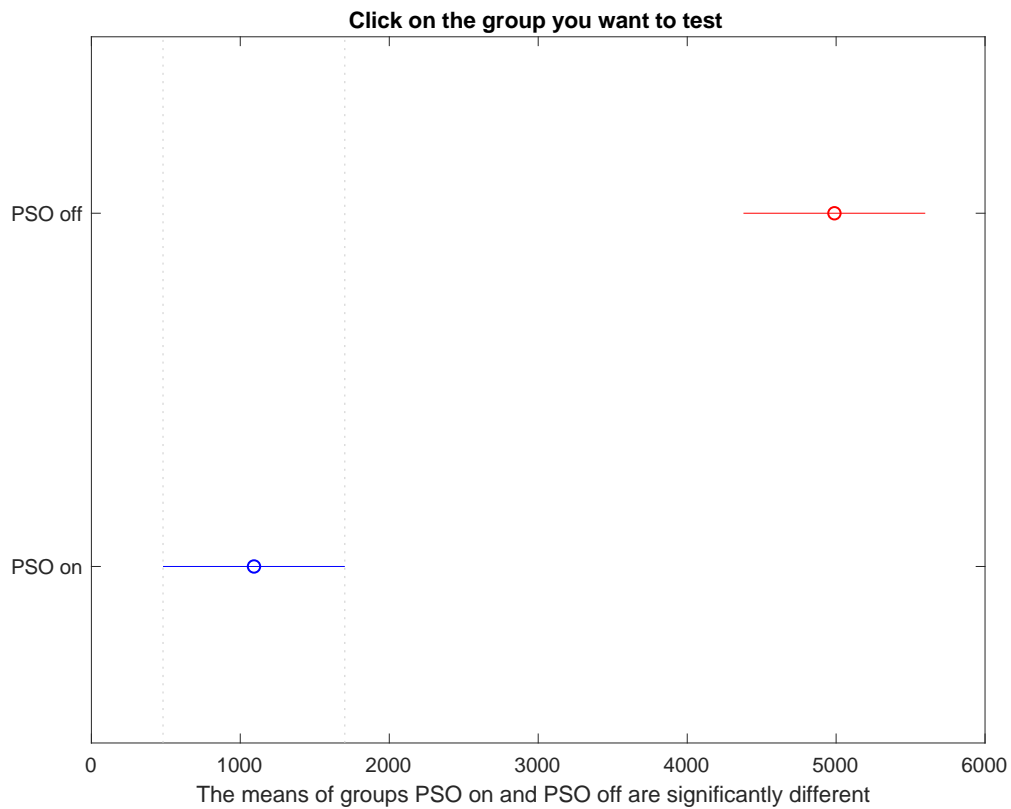


Figure 61. Testing for significant different result.

The simulation data presented show that the introduction of PSO guidance results in a significant reduction in the number of iterations needed to conclude the battle.

As can be seen, the addition of the PSO guidance definitely leads to a faster location of all known Red agents. At the same time, simulations have not revealed any significant impact on the number of casualties for both sides because PSO guidance by itself does not change the engagement sequence or probability of kill.

B. EFFECTS OF VARYING DETECTION RANGE

The value of the detection range, $d_d^{B \rightarrow R}$, obviously plays a major role in the success of the mission because it defines the LVC-PSO guidance switching moment. Figure 62 and 63 show the effect of varying $d_d^{B \rightarrow R}$ on the number of iterations needed to end the engagement and the number of casualties for both Blue and Red forces, respectively, at the end of the battle.

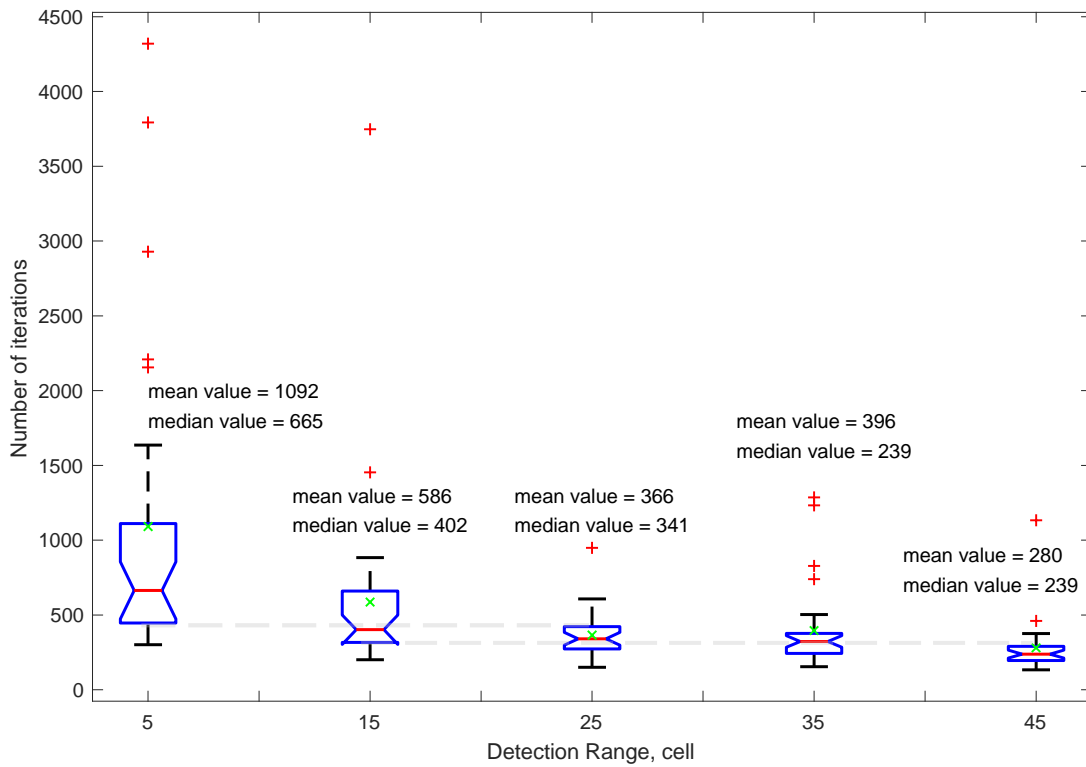


Figure 62. Number of iterations corresponding to various detection ranges.

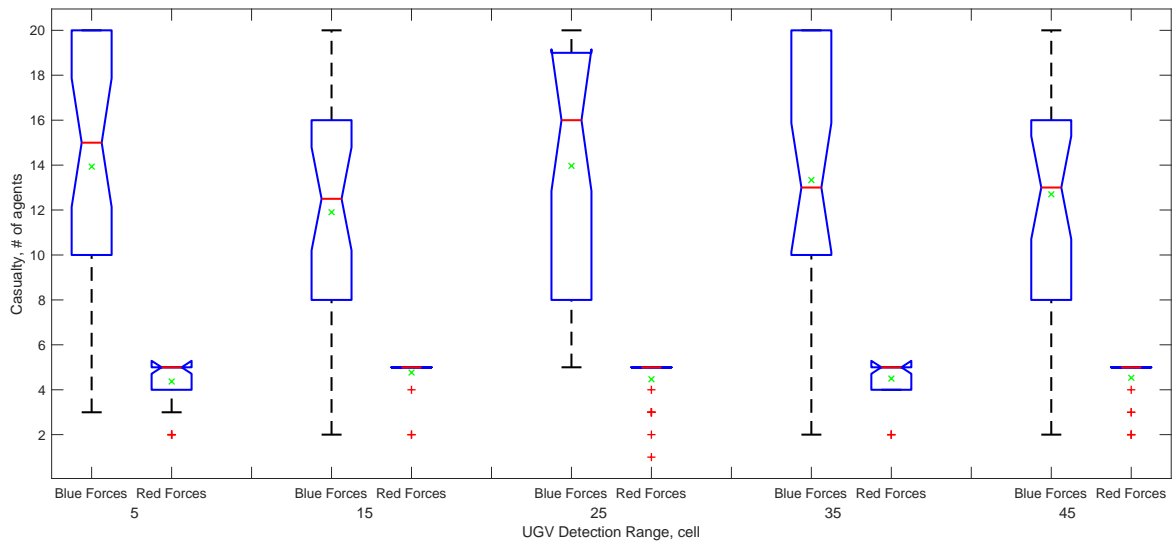


Figure 63. Number of casualties corresponding to various detection ranges.

Obviously, the number of iterations required to end the engagement decreases as the detection range increases. A small detection range means that the probability of losing track of a Red agent after it has been found increases. With a larger detection range, the Blue agents are able to start tracking Red agents earlier and thus longer, giving the rest of the Blue forces time to swarm towards the target.

Figure 63 demonstrates a significant difference between a detection range of five and 25 cells, after which the improvement starts reaching its saturation point (for a specific 100-by-100 grid setup and swarm size). Similar to the reasons for adding PSO guidance described the previous section, varying detection range does not seem to impact the number of casualties.

C. EFFECTS OF THE HOLONOMICITY CONSTRAINT DURING TRACKING

The findings of the previous chapter suggest that non-holonomic drive encourages exploration, thus increasing the coverage area. In this section, we explore the effects of holonomic drive during tracking with PSO guidance.

Figure 64 shows the effects of non-holonomic drive on the left and holonomic drive on the right during PSO guidance.

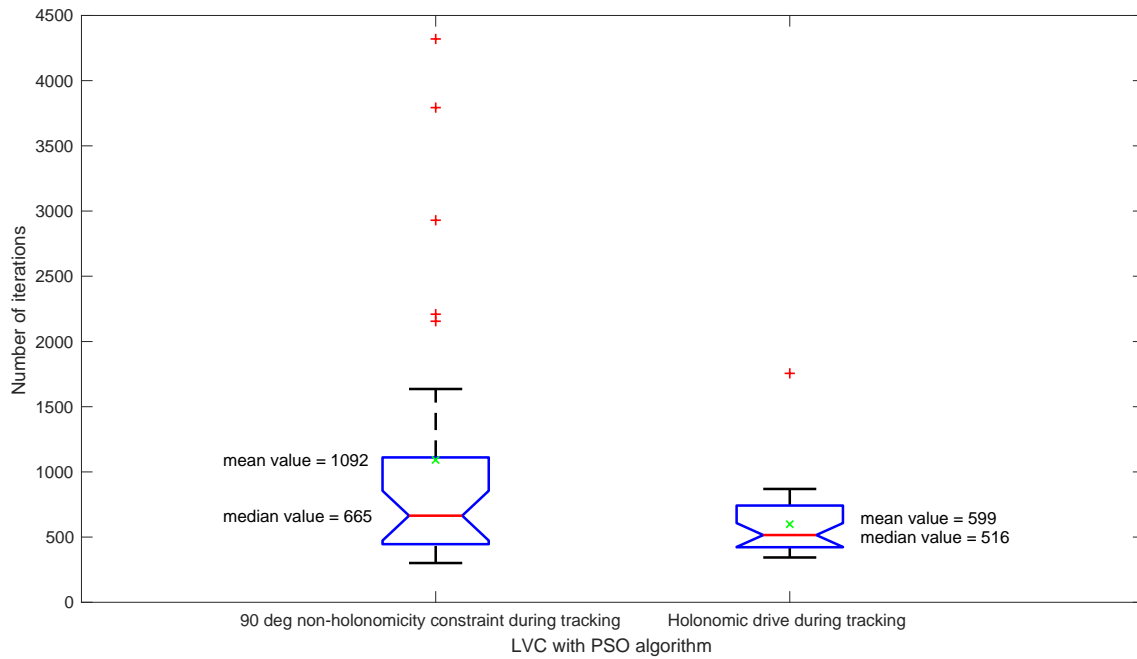


Figure 64. Comparison of holonomic and non-holonomic drive during tracking phase under PSO guidance.

Although non-holonomic constraint limits the Blue force’s agility (Figure 65 and 66), it leads to only a slight increase in the number of iterations needed to end the engagement. This is due to the fact that during tracking, holonomic drive would give the Blue agents the freedom to reach the targeted cell more quickly, as seen in Figure 64; however, the benefits did not lead to any significant reduction of the number of iterations required for the entire battle.

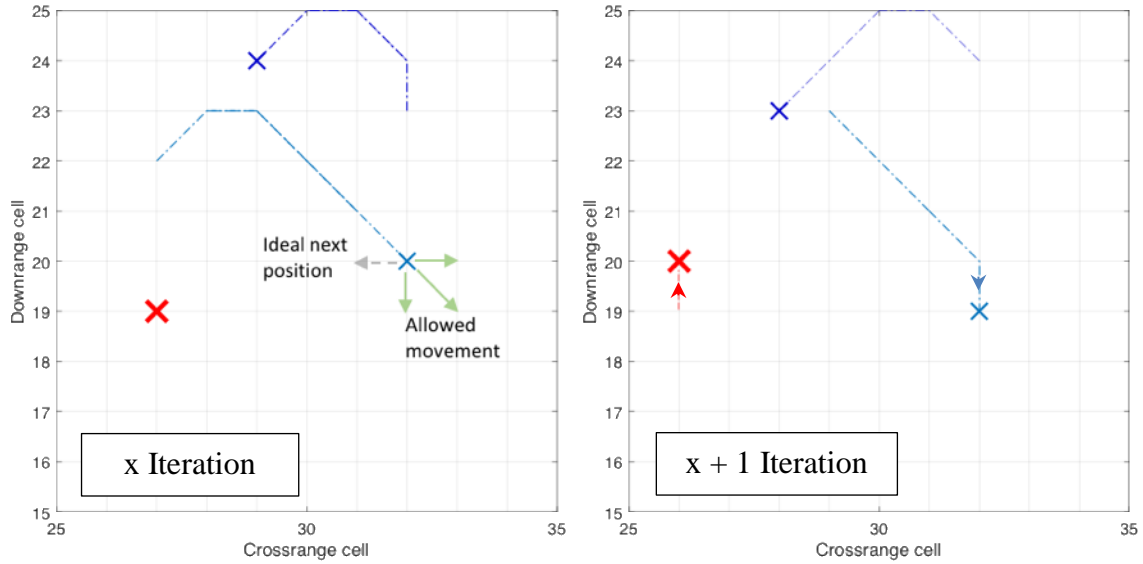


Figure 65. Snapshot of two consecutive iterations during tracking phase with a 90-degree non-holonomicity constraint.

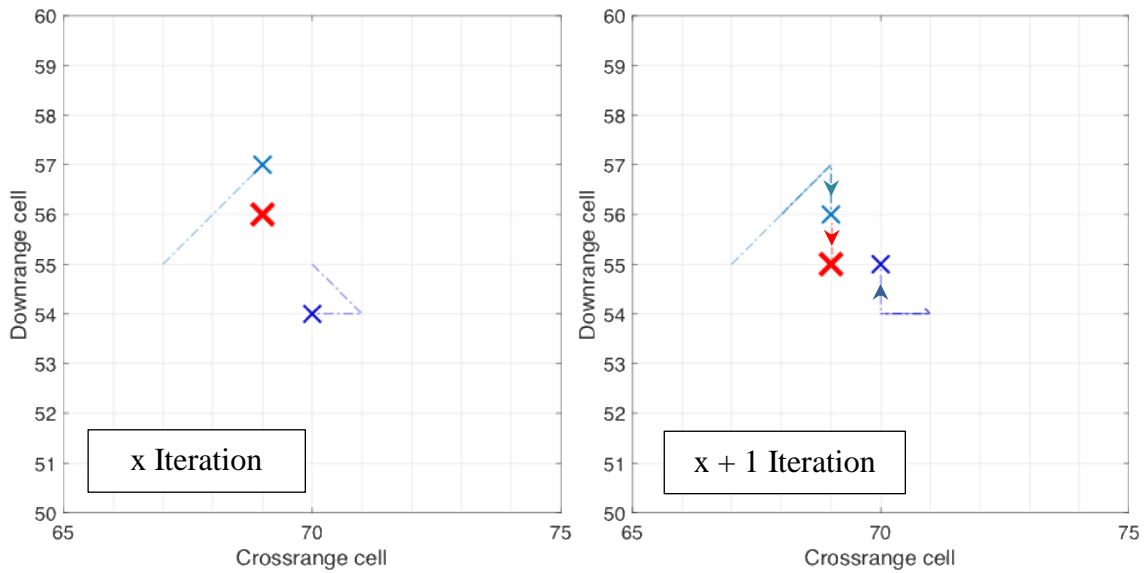


Figure 66. Snapshot of two consecutive iterations during tracking phase with holonomic drive.

D. EFFECTS OF PROBABILITY OF KILL

This section studies the engage phase. Specifically, it examines the probability of kill (P_k) without PSO guidance, which reflects the offensive capability. The effects of varying P_k for both forces is investigated. Figure 67, 68 and 69 show the box plots for the effects on number of iterations required to end the engagement with a fixed enemy offensive capability of 0.1, 0.5, and 0.9 in each figure while varying the Blue force's offensive capability of 0.1, 0.5, and 0.9, respectively.

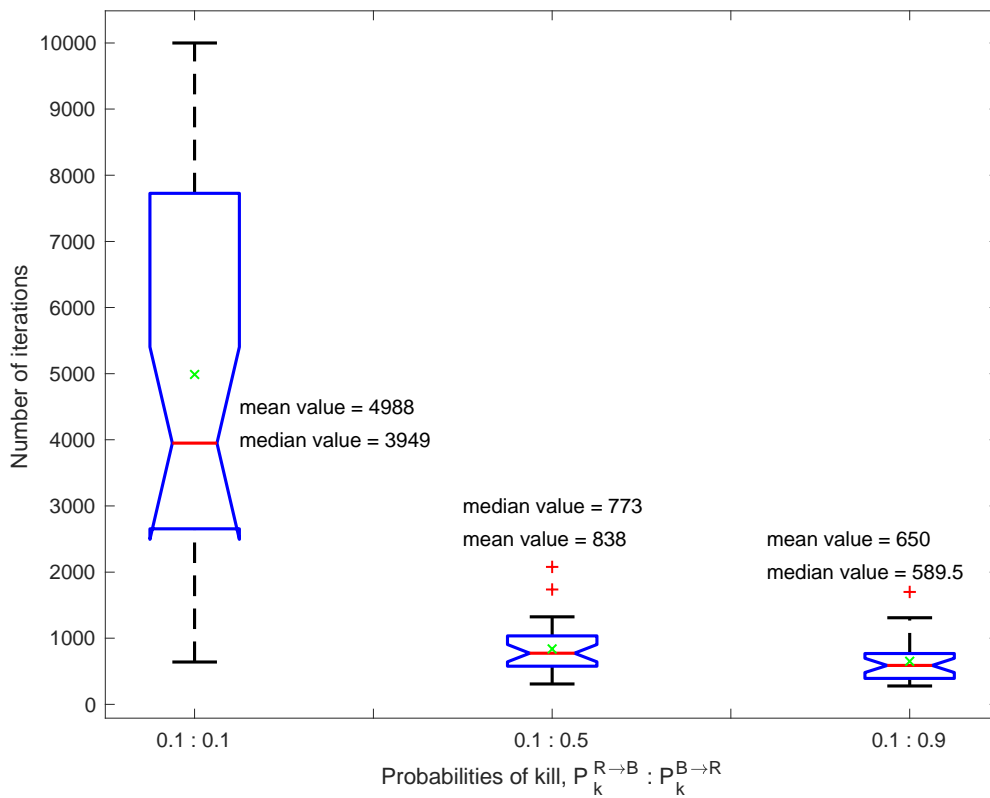


Figure 67. Effects of time with fixed enemy offensive capability of 0.1 and varying UGV agents' offensive capability.

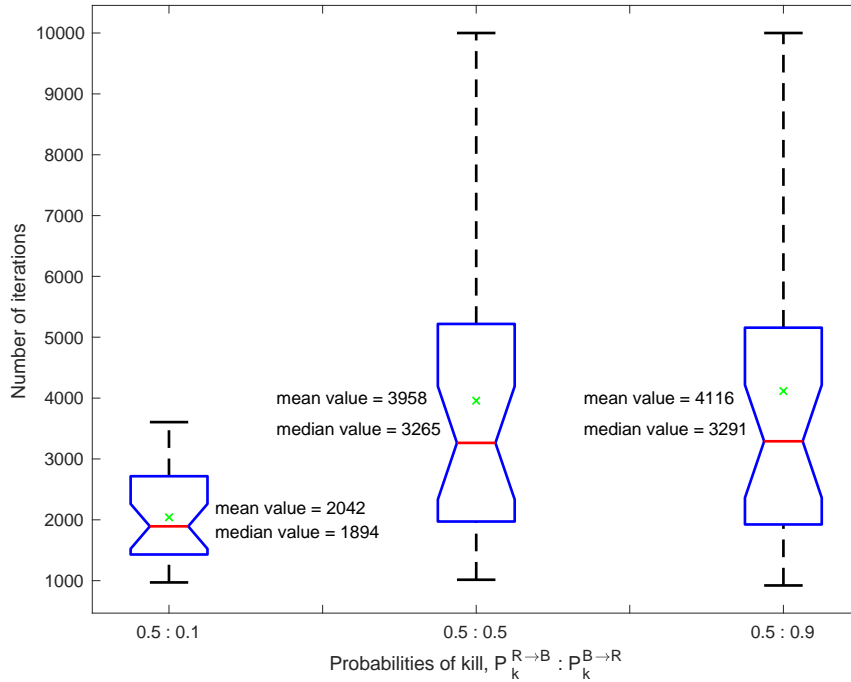


Figure 68. Effects of time with fixed enemy offensive capability of 0.5 and varying UGV agents' offensive capability.

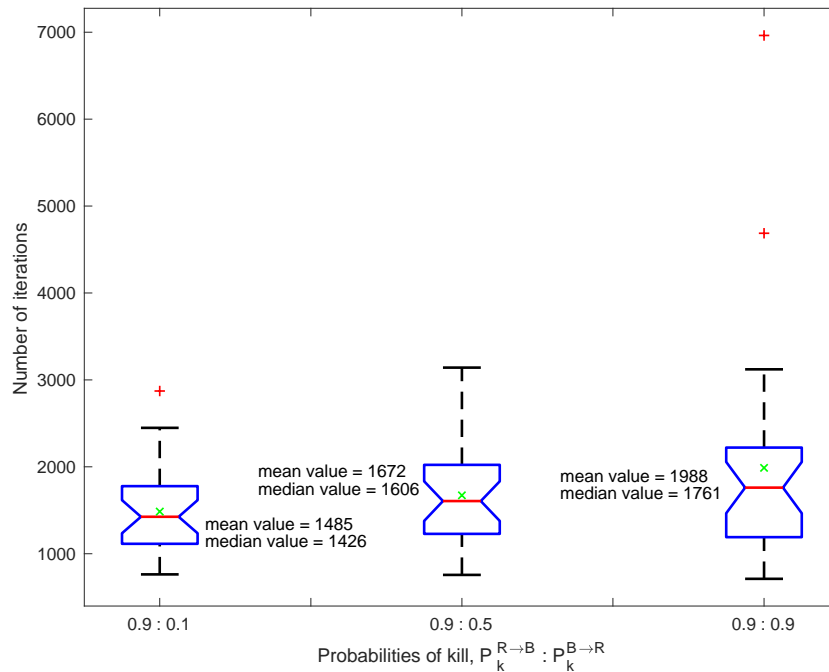


Figure 69. Effects of time with fixed enemy offensive capability of 0.9 and varying UGV agents' offensive capability.

Figure 67, 68 and 69 are complemented by Figure 70, 71 and 72 showing the number of casualties for both Blue and Red forces at the end of the battle.

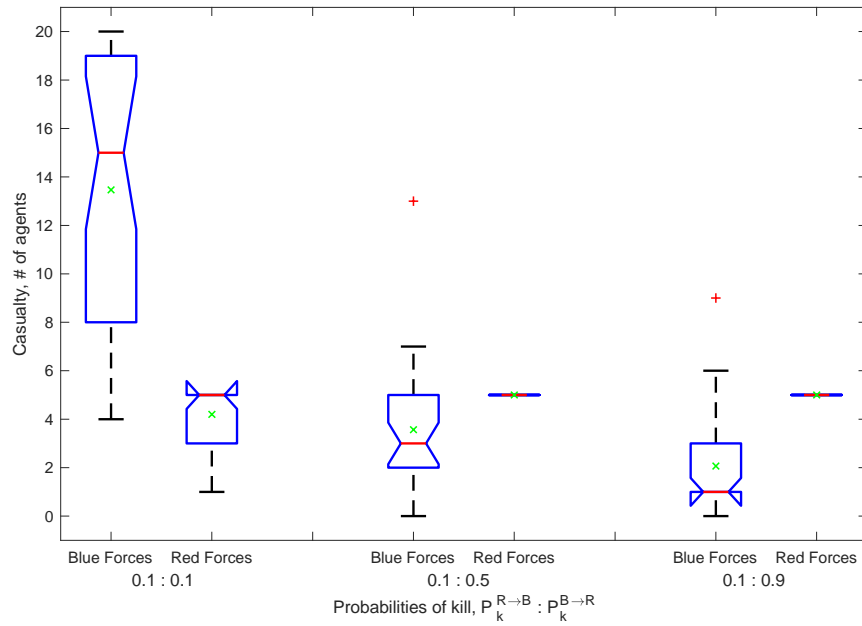


Figure 70. Effects on casualty rate with fixed enemy offensive capability of 0.1 and varying UGV agents' offensive capability.

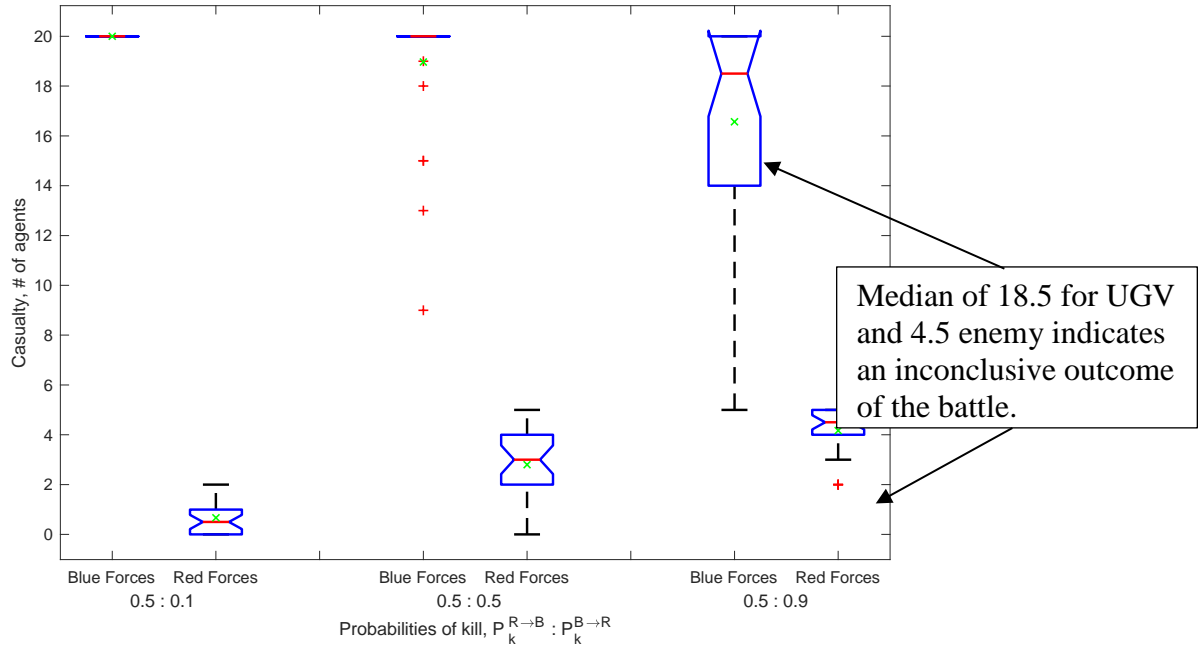


Figure 71. Effects on casualty rate with fixed enemy offensive capability of 0.5 and varying UGV agents' offensive capability.

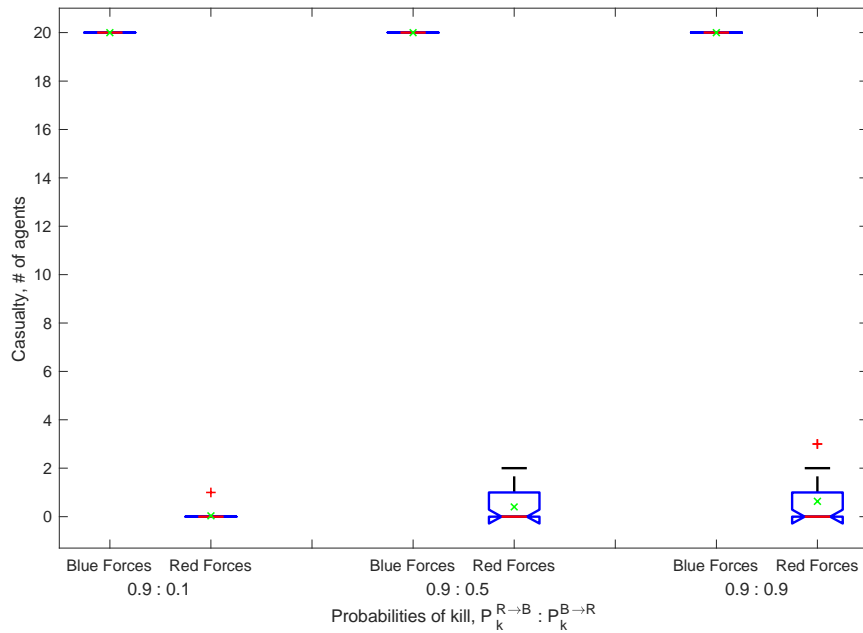


Figure 72. Effects on casualty rate with fixed enemy offensive capability of 0.9 and varying UGV agents' offensive capability.

There is no doubt that the values of $P_k^{B \rightarrow R}$ and $P_k^{R \rightarrow B}$ play a major role in the engagement outcome. Figure 67 shows that when the enemy offensive capability is low ($P_k^{R \rightarrow B} = 0.1$), the number of iterations changes significantly depending on the Blue force's offensive capability ($P_k^{B \rightarrow R}$). This is expected as the increase in the Blue force's offensive capability would kill the enemy faster, leading to the smaller number of iterations. This is confirmed by Figure 70, where the number of the Blue force's casualties decreases as its offensive capability increases. Figure 72 shows that with $P_k^{R \rightarrow B} = 0.9$ because of the tactical advantages given to the defending Red force (discussed in Section II.C), the Blue force has no chance of winning. If the value of $P_k^{R \rightarrow B} = 0.9$ is reduced to 0.5 (Figure 71) then the Blue force can possibly win with $P_k^{B \rightarrow R} > 0.5$, though suffering from heavy casualties. The small nominal values of probability to kill (0.1), therefore, were chosen to better demonstrate the effectiveness of swarming. As seen in Figure 70, the Red force starts winning even with equal probabilities to kill, but a significant increase of $P_k^{R \rightarrow B}$ is required to win unconditionally with a low casualty rate.

Table 5 shows the simulation result for the case of $P_k^{R \rightarrow B} = 0.5$ and $P_k^{B \rightarrow R} = 0.9$, where we observed an inconclusive outcome.

Table 5. Simulation results for $P_k^{R \rightarrow B} = 0.5$ and $P_k^{B \rightarrow R} = 0.9$.

Run	UGV Survivors	Enemy Survivors	Number of Iterations Taken	Winning Force
1	9	0	921	Blue
2	3	0	6,444	Blue
3	8	0	2,243	Blue
4	12	0	1,416	Blue
5	5	0	4,660	Blue
6	1	1	10,000	Red
7	7	0	1,656	Blue
8	5	0	3,383	Blue
9	0	1	9,600	Red
10	4	0	4,956	Blue
11	7	0	1,924	Blue

Run	UGV Survivors	Enemy Survivors	Number of Iterations Taken	Winning Force
12	0	3	2,069	Red
13	0	2	2,319	Red
14	0	1	7,872	Red
15	0	3	2,251	Red
16	0	1	4,714	Red
17	6	0	1,677	Blue
18	0	2	1,758	Red
19	15	0	921	Blue
20	9	0	2,310	Blue
21	0	3	1,488	Red
22	6	0	2,123	Blue
23	0	1	5,158	Red
24	1	1	10,000	Red
25	0	3	3,198	Red
26	0	1	5,029	Red
27	2	0	4,111	Blue
28	1	0	5,816	Blue
29	2	1	10,000	Blue
30	0	1	3,464	Red

Table 6. Summary of results for $P_k^{R \rightarrow B} = 0.5$ and $P_k^{B \rightarrow R} = 0.9$.

	Number of wins	Percentage of wins
UGV win	16	53.33%
Enemy win	14	46.67%

Table 6 shows a stand-still result where there is almost an equal probability of either the Red or the Blue force emerging victorious. For this specific context, the ratio of approximately 0.5 to 0.9 probability of kill results in about equal chances to win.

Based on these dynamics, the results presented in Figure 62 can now be explained even further. Being able to swarm towards the target earlier means that even with a small individual probability to kill, more agents taking a shot at the target increases the overall

probability of success to $1 - (1 - P_k^{B \rightarrow R})^m$, where m is the number of attacking agents. For example, with a nominal value of $P_k^{B \rightarrow R} = 0.1$ and $m=5$ the chances to kill quadruple.

E. EFFECTS OF KILL DISTANCE

Kill distances $d_k^{B \rightarrow R}$ and $d_k^{R \rightarrow B}$ are defined as minimum distances required to engage (fire) at the opponent. Larger kill distance allows for engagement at a longer range, which provides an advantage over the opponent. As a baseline for all previous simulations, the Red force was assigned a kill distance of two cells compared to the Blue force with a kill distance of one cell (which was an advantage purposely given to the Red force because its defensive posture). Figure 73 and 74 show the effect of varying the $d_k^{B \rightarrow R} / d_k^{R \rightarrow B}$ ratio while assuming $P_k^{B \rightarrow R} = P_k^{R \rightarrow B} = 0.5$.

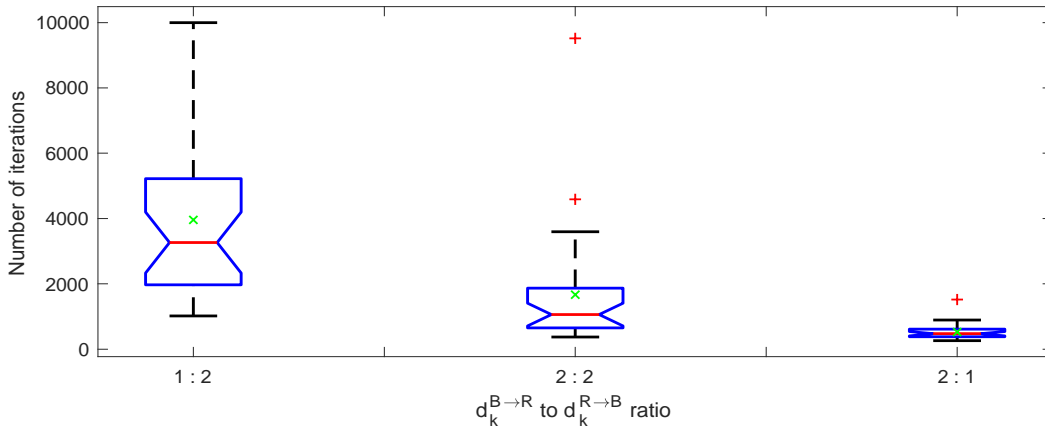


Figure 73. Effects of kill distance on number of iterations.

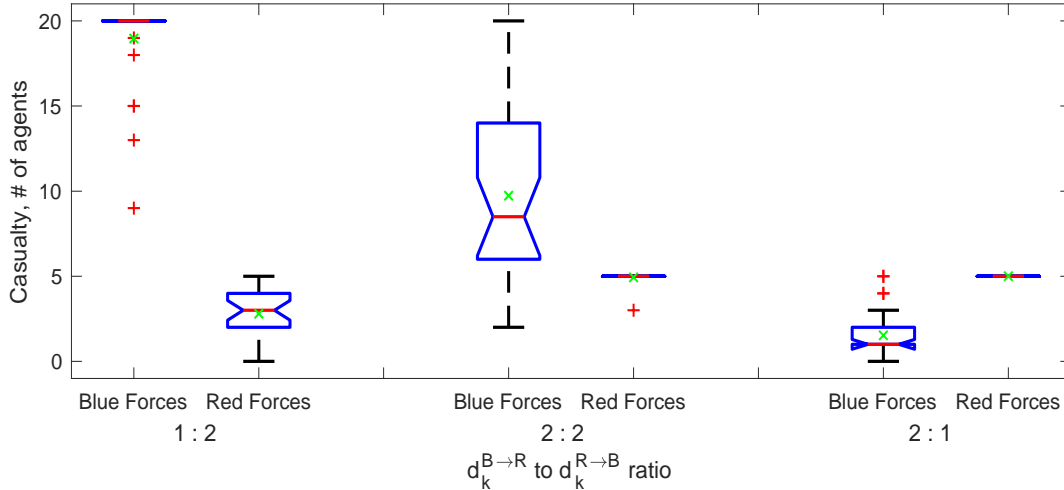


Figure 74. Effects of kill distance on number of casualties.

As seen, the baseline situation (1:2 ratio) results in the high number of iterations to end the engagement (Figure 73) and leads to the Blue force's defeat (Figure 74). Relaxing this ratio to 2:2 decreases the number of iterations and changes the outcome of engagement. The 2:1 ratio results in the smallest number of iterations and the unconditional win of the Red force.

F. EFFECTS OF KILL SEQUENCE

Continuing the results from simulations of the previous section, the effects of change in kill sequence to allow the Blue force to fire first instead of the Red force was investigated; thus, giving the Blue force the advantage. In previous sections, the ability to engage first is an advantage given to the Red force due to its defensive posture. The box plot in Figure 75 shows the effect on the number of iterations required to end the engagement for the same three kill distance setups from the previous section.

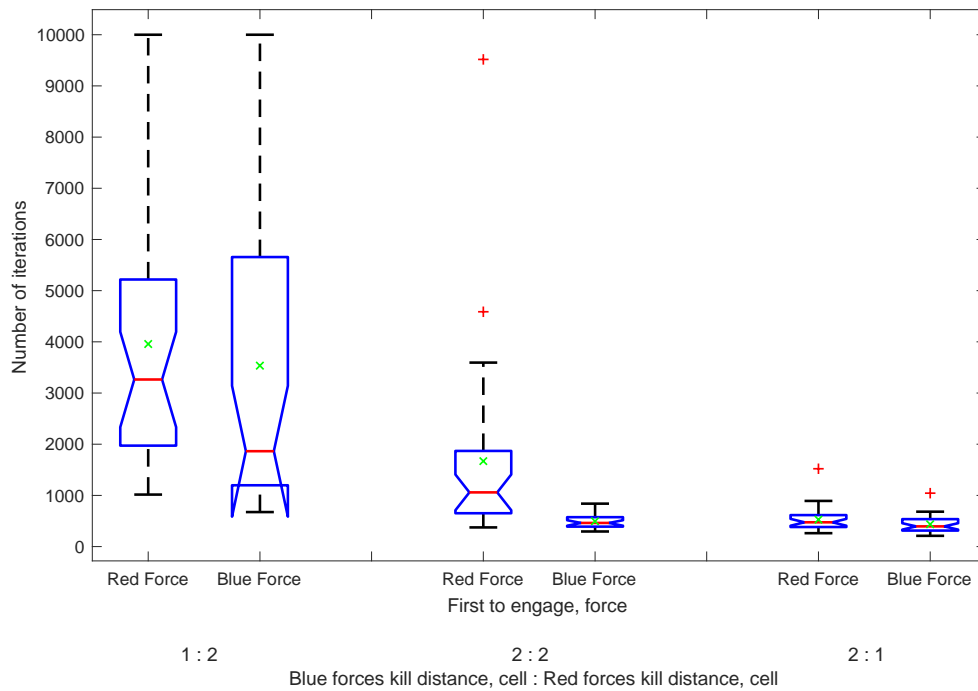


Figure 75. Effects of kill sequence on the number of iterations.

The box plots in Figure 74 and 75 show the effect on the number of Blue force casualties and Red force casualties at the end the engagement for the same three kill distance setups presented in the previous section.

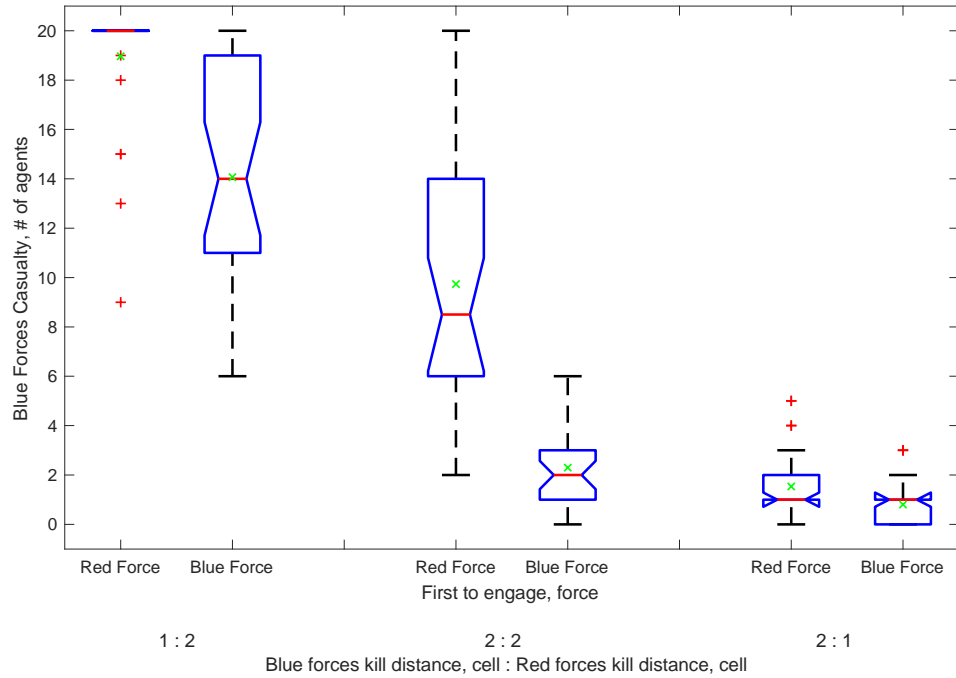


Figure 76. Effects of kill sequence on the number of Blue force casualties.

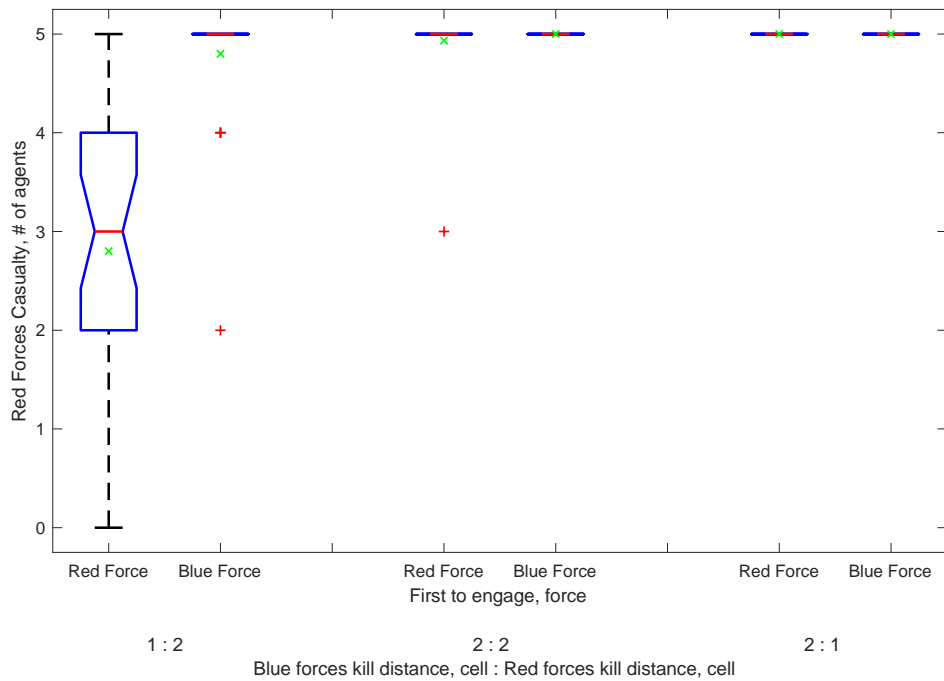


Figure 77. Effects of kill sequence on the number of Red force casualties.

The results shows that for cases when the Blue force is not at an advantage in kill distance (plots on the left and center), the change in kill sequence gives the Blue force the ability to engage first, allowing the Blue force to sustain fewer casualties and increasing the win rate for the Blue force. In a case where the Blue force already gains an advantage in kill distance (on the right), the added advantage in kill sequence does not further improve the number of iterations or the number of Blue force casualties.

G. URBAN OUTDOOR ENGAGEMENTS

The effects of PSO guidance and detection range on outdoor urban operations is studied in the section. Figure 78 shows the starting configuration of both Blue and Red forces.

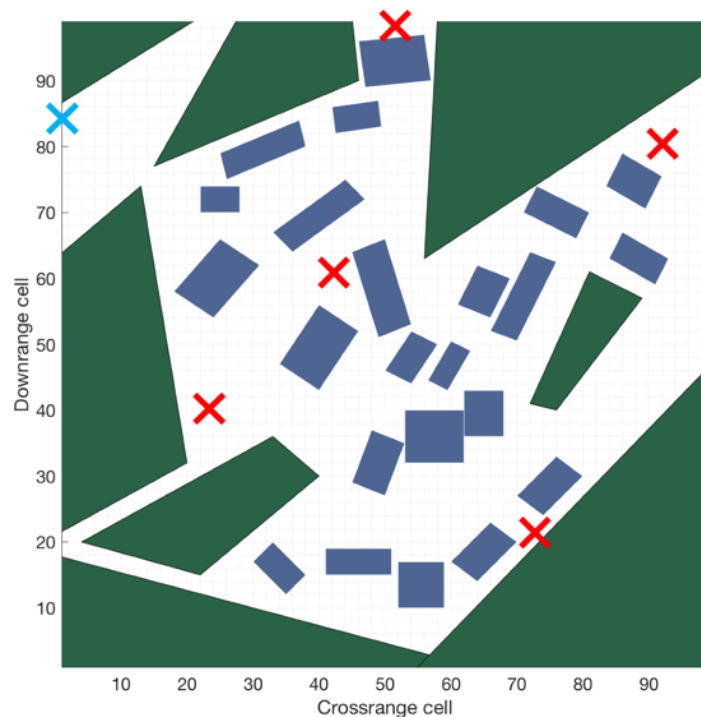


Figure 78. Starting configuration for outdoor urban operation.

The use of PSO guidance in outdoor urban operations demonstrates tendencies somewhat similar to those of the open-space engagement. To this end, the following figures

demonstrate the effect of varying the detection range with and without PSO guidance on the number of iterations (Figure 79) and number of casualties (Figure 80).

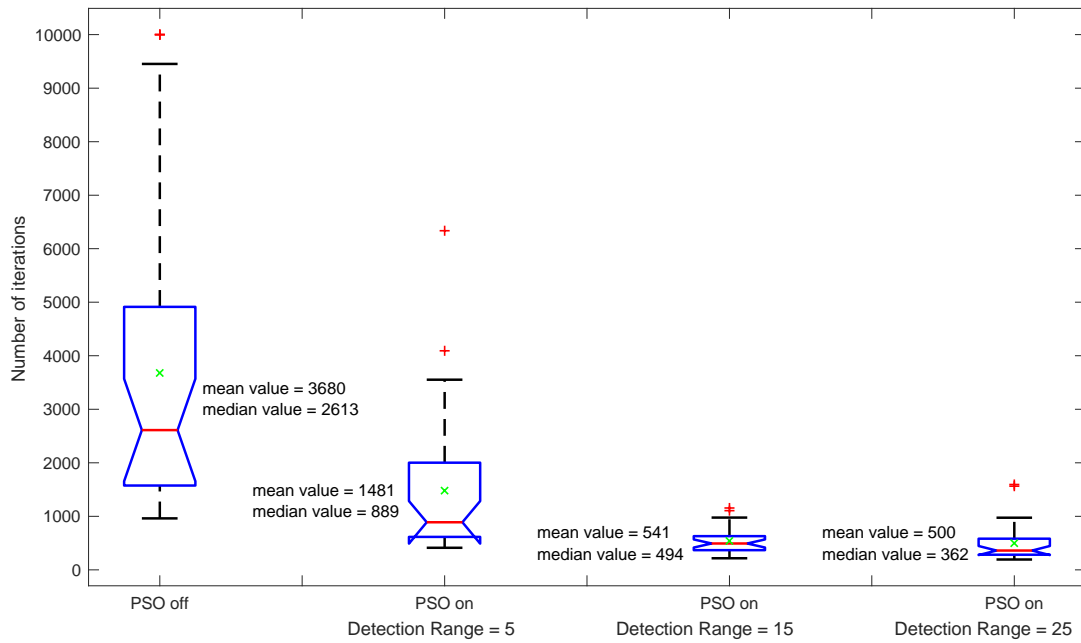


Figure 79. Effects on number of iterations with and without PSO guidance for outdoor operation.

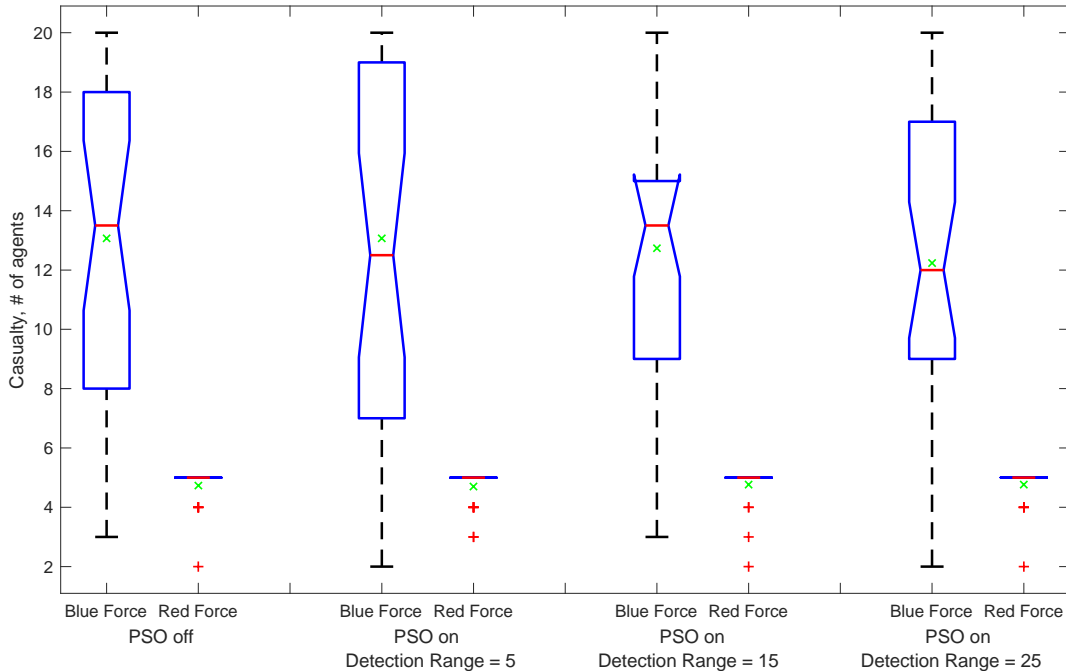


Figure 80. Effects on number of casualties with and without PSO guidance for outdoor operation.

The tendency to decrease the number of iterations required to end the engagement with an increase in the detection range holds. The effect of switching to PSO guidance is also positive (Figure 79). However, the introduction of PSO guidance at the detection range of five cells seems to have little effect on the number of casualties (Figure 80).

This study concludes that it is beneficial for the Blue force to have as wide a detection range as possible so as to locate the enemy and trigger the PSO algorithm in a shorter time. However, further enhancement of detection sensors might entail technical and cost challenges. Furthermore, the dampening of the detection sensors due to the urban environment itself have not been considered in this study but would increase its complexity and benefits.

H. INDOOR ENGAGEMENTS

In this section, the effects of PSO guidance on indoor operation is studied. Figure 81 shows the starting configuration of both the Blue and Red forces.

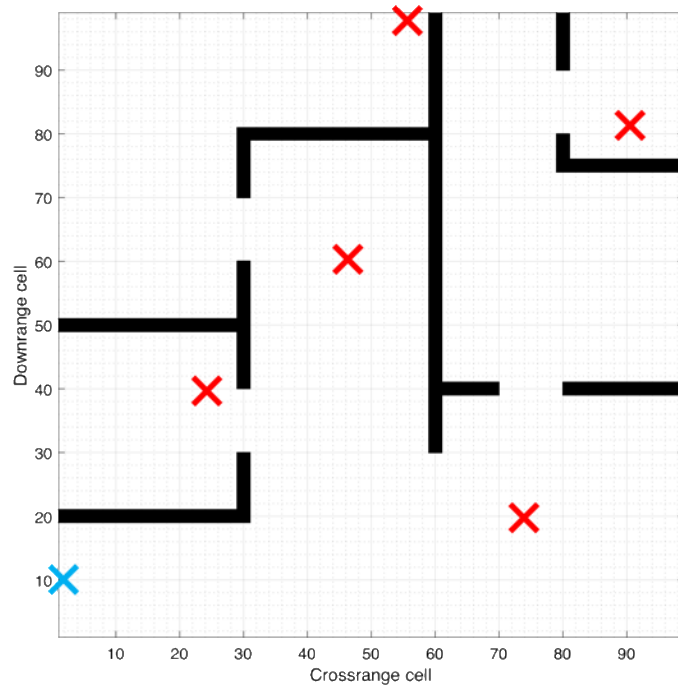


Figure 81. Starting configurations for indoor operation.

The effects on the number of iterations and casualties corresponding to various detection ranges with and without PSO guidance can be seen in Figure 82 and 83.

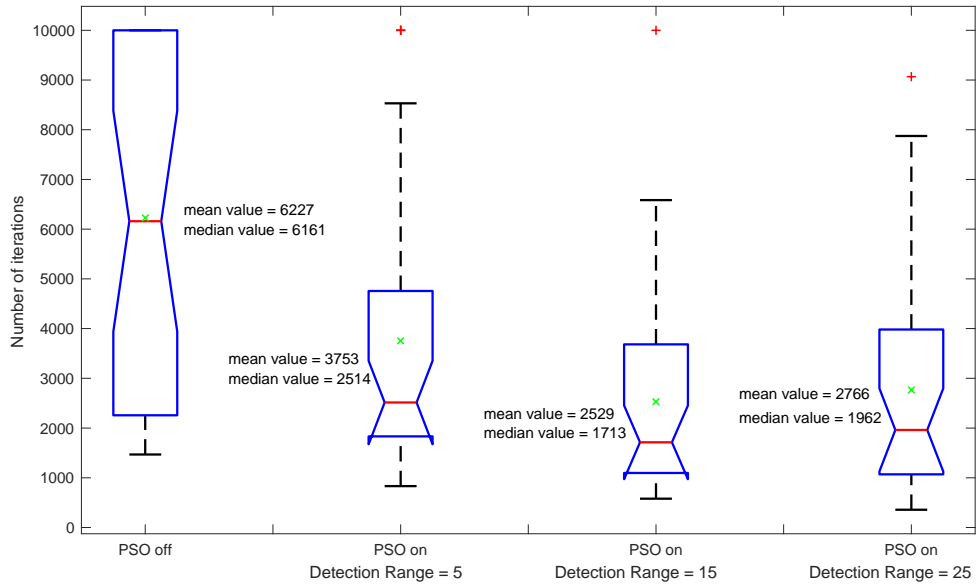


Figure 82. Effects on number of iterations, with and without PSO guidance, for indoor operation.

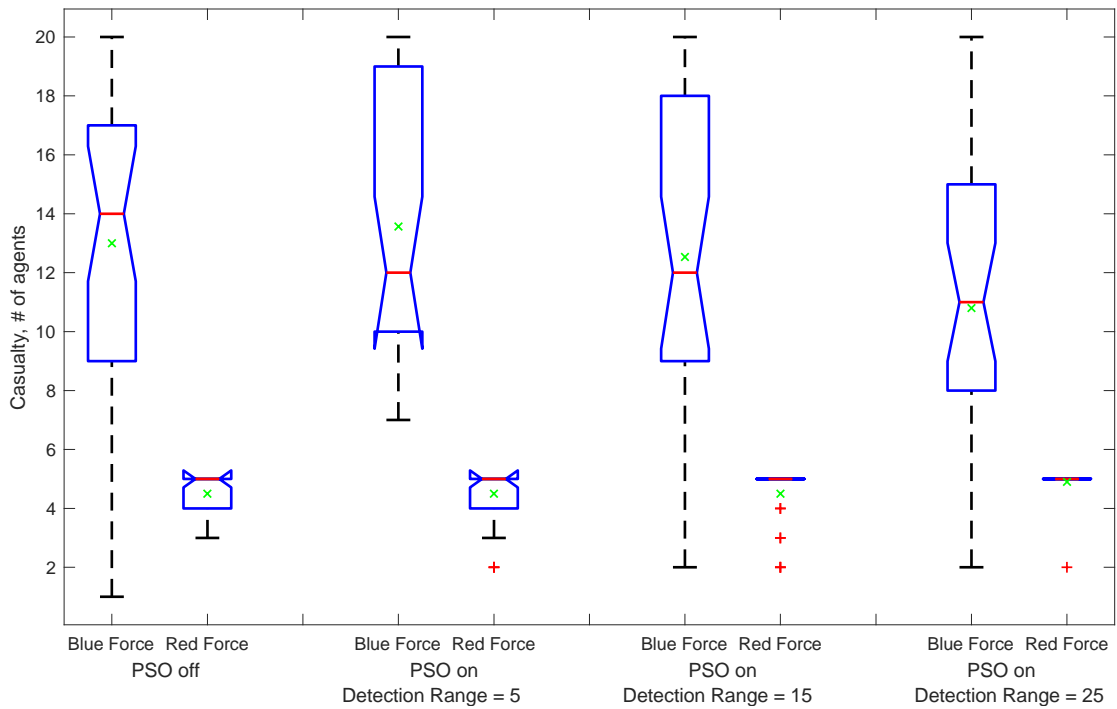


Figure 83. Effects on number of casualties, with and without PSO guidance, for indoor operation.

The tendencies revealed for the urban outdoor engagements do not exactly match those of indoor engagements (compare Figure 82 and 83 with Figure 79 and 80).

A closer look allows one to determine that this is due to the inability of the Blue force's agents to avoid concave obstacles typical for the indoor environment. As seen from illustrations presented in Figure 84, the increase in detection range causes the Blue agents to employ PSO guidance earlier and longer, thus fixing on the shortest path to the target and getting stuck behind an obstacle (on the right). With the smaller detection range (on the left), the probability of reverting from PSO guidance to LVC guidance is higher, thus allowing the Blue agents to possibly maneuver to a position where there are no obstructions before PSO guidance is triggered again.

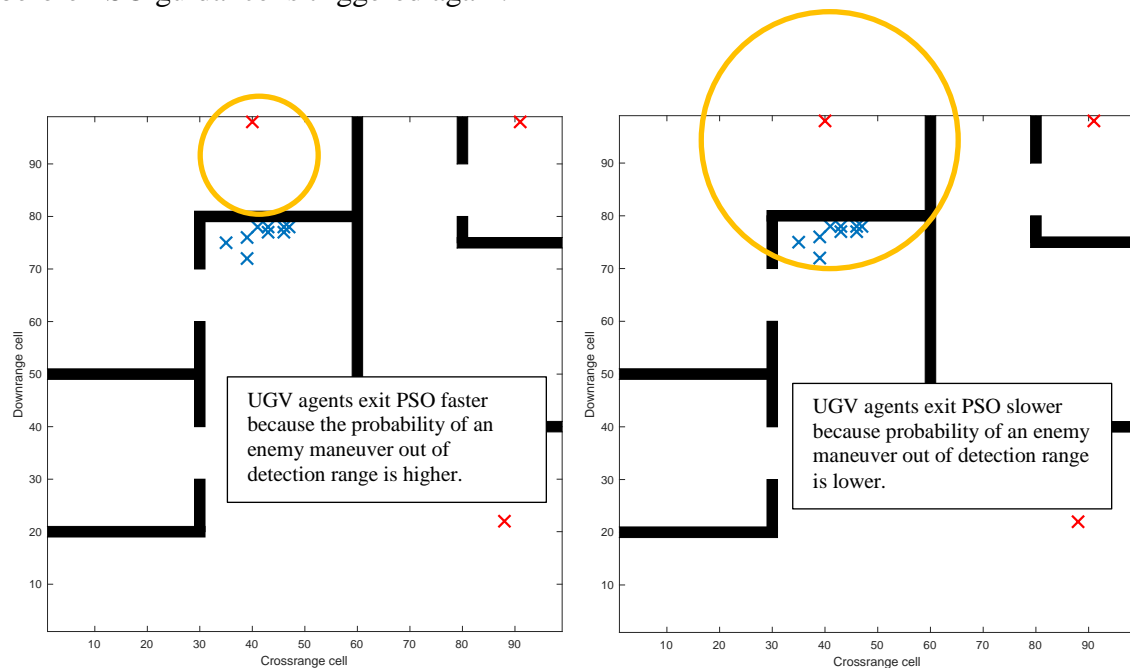


Figure 84. Inability to avoid obstacles with low detection range (left) and high detection range (right).

Being stuck behind an obstacle is a known problem for almost any algorithm, and as a result certain remedies allowing addressing it have been developed already (Wang et al. 2018). It should be noted though that for this particular application the Blue agents may get 'unstuck' by themselves even without any additional measures undertaken. The three opportunities include the following:

- Red agent maneuvering out of Blue agents' detection range, causing the latter to exit the PSO guidance phase.
- Red agent being destroyed by other Blue agents that approached it via a different path, thus allowing stuck agents to exit the PSO guidance phase.
- Another Red agent closer to the stuck Blue agent is detected, thus triggering a change of \mathbf{G}_i^{best} .

While this problem could possibly happen in an outdoor operation as well, the probability that a Blue agent gets “stuck” is lower due to the different construct of obstacles in both operations. Most Blue agents get “stuck” behind an indoor obstacle because of its concave features. By contrast, in the outdoor operation most obstacles have convex features and thus allow the Blue agent to maneuver past them much more easily and quickly when the detected Red agent maneuvers slightly.

I. EFFECTIVENESS OF ALVC GUIDANCE

This section studies the effects of introducing ALVC guidance and then compares those to the effects of PSO guidance, or lack thereof. Figure 85 shows the box plots, comparing the LVC and ALVC algorithms, with and without PSO guidance, and the effects on the number of iterations required to end an engagement. The effects on casualties of the LVC and ALVC algorithms, with and without PSO, are shown in Figure 86.

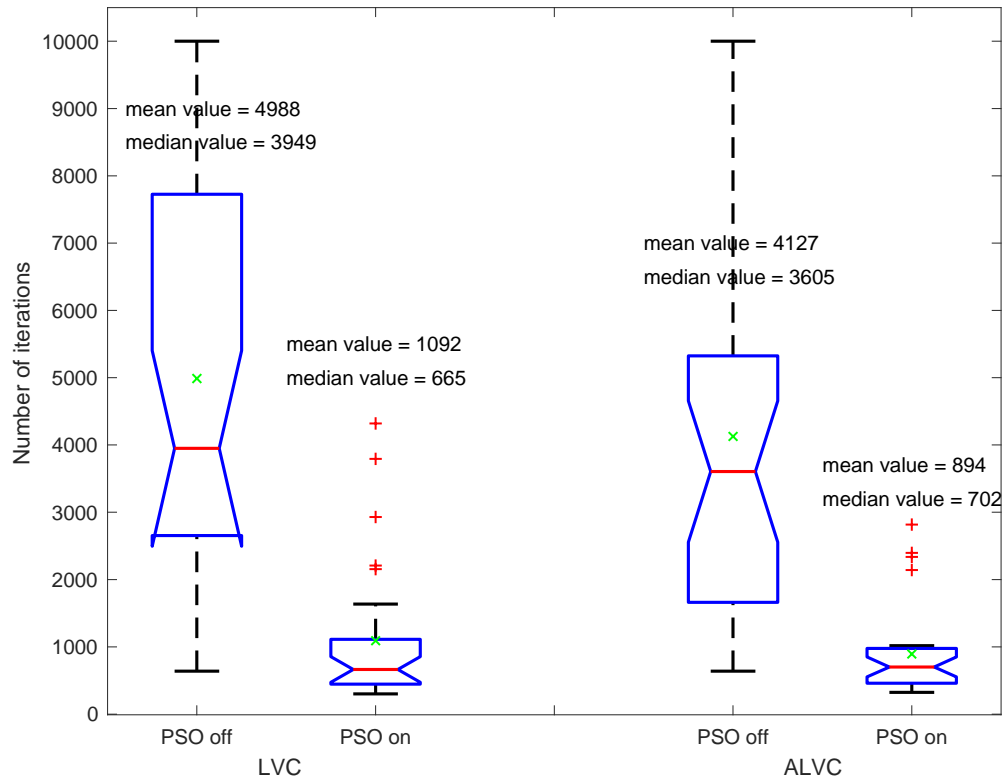


Figure 85. Effects of LVC with ALVC algorithms, with and without PSO guidance, on number of iterations.

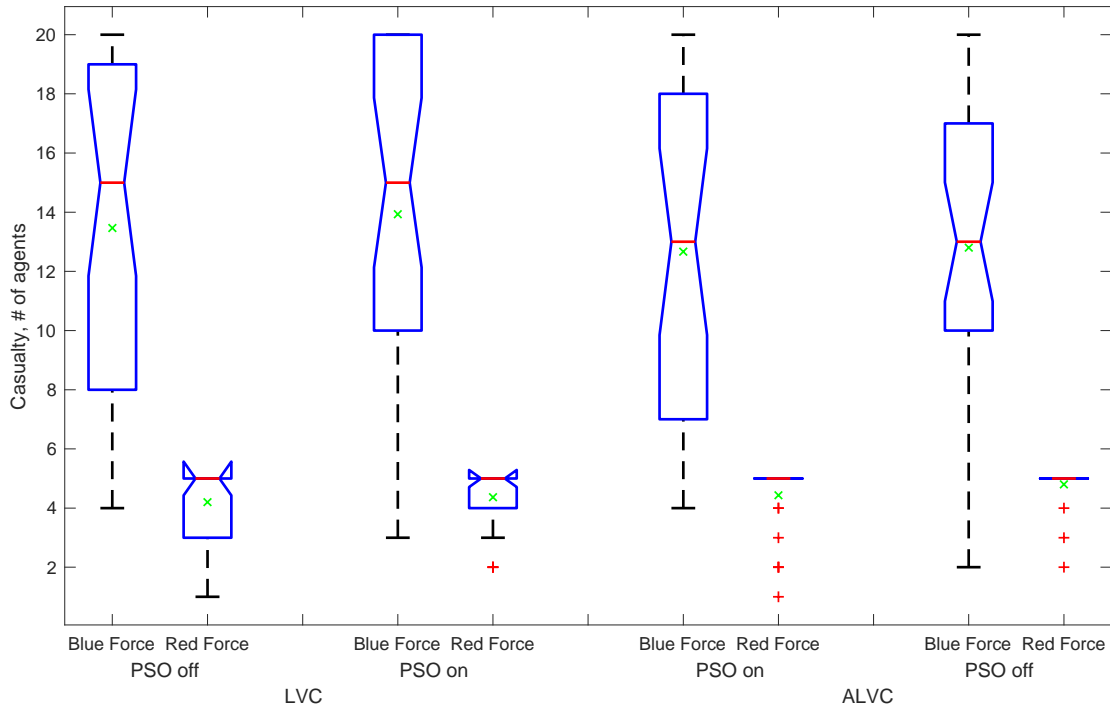


Figure 86. Effects of LVC with ALVC algorithms, with and without PSO guidance, on number of casualties.

The introduction of PSO guidance for both algorithms improved the number of iterations required to end the engagement. This supports the previous conclusion that the introduction of PSO guidance reduces the number of iterations. However, there are no significant improvements when the ALVC algorithm is compared to the LVC algorithm. While the ALVC algorithm improves area coverage, as shown in the previous chapter, it does not aid in an operation where searching, tracking, and engaging is the priority. There is no significant effect on the number of casualties for both the Blue and Red forces with the introduction of the ALVC algorithm. This result is somewhat expected as ALVC does not change the sequence or probability of kill and thus should not affect the battle outcome.

J. LIMITATIONS OF ALVC GUIDANCE

A discovered limitation of the ALVC algorithm was that the Blue agents were not programmed to avoid obstacles and thus would always choose the shortest path to reach their goal. This resulted in the Blue agents being stuck behind the obstacles and unable to proceed as shown in Figure 87.

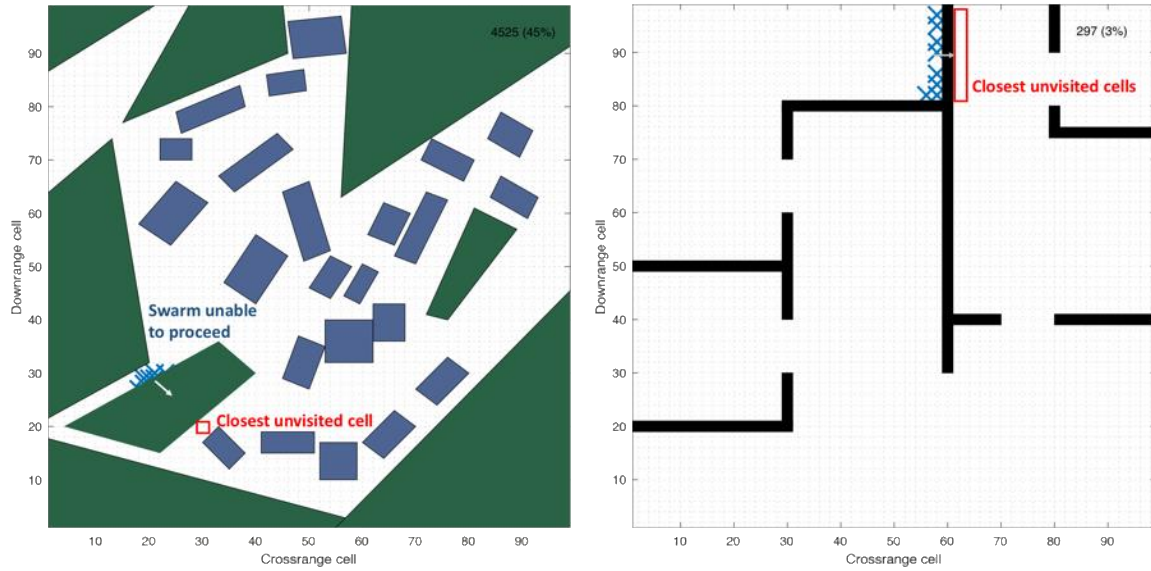


Figure 87. Limitations of the improved LVC algorithm in urban and indoor operations.

On the surface, the inability to avoid obstacles seems similar to the limitation found for PSO guidance in indoor operations (Figure 84). On further investigation, however, the problem with the ALVC algorithm is more severe. This is because it is impossible for the Blue agents to maneuver away once they are ‘stuck’ behind an obstacle because the target (closest unvisited cell) is stationary compared to the previous case where the target (Red agent) is moving. In that case, the target’s movement allowed the possibility of a change in shortest path.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

A. SUMMARY

This paper presented and evaluated guidance algorithms for a UGV swarm operating in the urban environment, using MATLAB for simulations. The mission of the UGVs is broken down into two phases. The first phase is identified as the search phase and its measure of effectiveness is area coverage. The second phase is the track and engage phase and its measures of effectiveness are the time (the number of iterations) required to end an engagement as well as number of casualties for the Blue and Red forces. A summary of the investigated algorithms and input parameters is shown in Table 7.

Table 7. Summary of algorithms and input parameters investigated.

Phase	Algorithm	Input Parameters
Search	LVC	Number of UGV agents
	LVC	Maximum number of iterations
	LVC	Starting configuration
	LVC	Collision avoidance constraints
	LVC, ALVC	Non-holonomicity constraints
	LVC, ALVC	Outdoor and indoor urban environments
Track and Engage	LVC, PSO	Non-holonomicity constraints
	LVC, PSO	Detection range
	LVC, PSO	Probability of kill
	LVC, PSO	Kill distance
	LVC, PSO	Kill sequence
	LVC, ALVC, PSO	Outdoor and indoor urban environments

B. MAIN FINDINGS

This section attempts to answer the two research questions introduced in Chapter I.

1. Are the algorithms developed suitable for the swarm UGVs to achieve their mission?

In the developed simulation environment, it was shown that employing the LVC guidance algorithm during the area search phase of the mission works well for the open-space and urban (both outdoor and indoor) operational environments. The addition of PSO-

based guidance at the track and engage phase has a positive effect, resulting in about a fivefold reduction in the time required to locate and destroy all known targets. Since PSO guidance does not change the engagement sequence or probability of kill explicitly, it seems to have little effect on the number of casualties of the attacking side. Furthermore, in indoor operations featuring concave obstacles, PSO guidance needs more improvement so that Blue agents avoid being stuck behind an obstacle with no way out.

2. What are the factors that affect the UGV swarm's ability to achieve its mission?

First, increasing the number of UGVs in the swarm would assist in locating targets in a shorter amount of time. Nonetheless, there is a saturation point beyond which any increase would result in diminishing returns. Although increasing the number UGVs would also lead to a higher probability of win in the track and engage phase, the number of UGVs deployed might be constrained by budget and technology.

Secondly, multiple entry points into the operational area is beneficial by encouraging exploration, which in turn improves area coverage and thus allows the UGVs to locate the enemy in a shorter period of time. Multiple entry points, however, are not always possible due to terrain or approach constraints.

Third, the findings related to the non-holonomicity constraint show that non-holonomic drive improves area coverage and thus locating the targets in less time. The results also seem to suggest the benefits of non-holonomicity constraints are amplified when obstacles are present. While narrower constraints encourage exploration, which is ideal for area coverage, they seem to be a hindrance while the swarm is tracking a moving target as its degree of freedom is limited.

Fourth, increased detection range leads to better situational awareness for the UGVs and allows for earlier activation of PSO guidance, which reduces the total engagement time. On the other hand, increasing detection range might be challenging due to technological and budget constraints.

Finally, the three input parameters affecting the number of casualties are the probability of kill, kill distance, and kill sequence. These parameters in the operational

context refer to the combination of the ability of each UGV's sensors to shoot, the UGV's weapon range, and its ability to detect, respectively. Despite advancements in the technology and this field of research today, human sensors combined with cognitive abilities still prove superior to a machine in such a complex environment, and thus, to successfully meet their mission the UGVs must outnumber the humans.

C. RECOMMENDATIONS FOR FUTURE WORK

Future work is recommended to improve on the ALVC guidance algorithm so that it is able to overcome obstacles. This would allow for the comparison of the LVC and ALVC guidance algorithms and generate more insights.

It is also recommended that a weighted approach be implemented for LVC guidance. This approach would assign values to the cells depending on the number of visits made. As more UGV agents enter a cell, this value would increase. UGV agents are programmed to move to neighboring cells with the lowest value. This approach might help solve the limitations of the ALVC guidance algorithm and allow the UGV agents to overcome obstacles.

Varying the distance of collision avoidance constraints could also be further investigated. Intuitively, one can surmise that increasing the distance in the collision avoidance constraint might encourage exploration and hence improve area coverage. Nevertheless, similar to the effects of the holonomicity drive, an increase in the distance in the collision avoidance constraint might affect the tracking phase when the UGV agents swarm towards a target.

Lastly, it is recommended that further work reduce the number of cells for indoor operations for a more realistic simulation. The effects of the input parameters might be different in a reduced cell operational area.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. SEARCH PHASE WITH LVC GUIDANCE

```
for SS = 1000;

ml = 1;
for Mainloop = 1 : ml
close all

%% Defining swarm
% Neighboring cells numeration
%   4   3   2
%   5   X   1
%   6   7   8
N=SS; %number of iterations
SwarmSize = 20; %number of agents in a swarm
collisionavoidance = 1; %1 for on, 0 for off
holonomicity = 90; %360 for "off", 90, 180 270 degree for "ON"

% Swarm Starting position
Center = 0;
Cornertopright = 0;
Cornebttleftfttopright = 0;
Cornerallsides = 0;
Row = 0;
Btrightcorner = 0;
%for indoor/outdoor starting configuration
configuration = 1; %1 2 and 3 for outdoor, 4 for indoor

CelSz=1;           % cell size
GrSiz=99;          % grid size
A = zeros(1,8);
swarm=zeros(SwarmSize,9);

%choose map
outdoor = 1; % outdoor map, impossible city
indoor = 0; % indoor floorplan of one building
\

%% Defining buildings
if outdoor == 1,
run('Buildings_Obstacles.m')
else if indoor == 1,
run('indoor_floorplan.m')
end
end

%% Initial conditions

for s = 1:SwarmSize

if configuration == 1;
swarm(s,1) = 1;
swarm(s,2) = 85;
end

if configuration == 2;
if s <= (SwarmSize /2)
swarm(s,1) = 1;
swarm(s,2) = 85;
```

```

else
swarm(s,1) = (GrSiz);
swarm(s,2) = 85;
end
end

if configuration == 3;
if s <= (SwarmSize/3)
swarm(s,1) = (GrSiz);
swarm(s,2) = 85;
else if s > (SwarmSize /3) & s <= 2*(SwarmSize /3)
swarm(s,1) = 1;
swarm(s,2) = 20;
else if s > (2*(SwarmSize /3))
swarm(s,1) = 1;
swarm(s,2) = 85;
end
end
end
end

if Center == 1;
swarm(s,1) = (GrSiz+1)/2;
swarm(s,2) = (GrSiz+1)/2;
end

if Bttrightcorner == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
end

if Cornertopright == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
end

if Cornerallsides == 1;
if s <= (SwarmSize/4)
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
else if s > (SwarmSize /4) & s <= 2*(SwarmSize /4)
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
else if s > (2*(SwarmSize /4)) & s <= 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = (GrSiz);
else if s > 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = 1;
end
end
end
end
end

if Row == 1 ;
swarm(s,1) = round(((GrSiz)/SwarmSize ) * s) ;
swarm(s,2) = 1;
end

if Cornebttlefttopright == 1;
if s <= (SwarmSize /2)
swarm(s,1) = 1;

```

```

    swarm(s,2) = 1;
    else
    swarm(s,1) = (GrSiz);
    swarm(s,2) = (GrSiz);
    end
end

end

swarm(:,5) = 0; %initial x transition
swarm(:,6) = 0; %initial y transition

%% Building block calucations
if indoor == 1 | outdoor ==1
buildings = size(blowerleft,1);

for bb = 1:buildings;
bupperg(bb) = (bupperright(bb,2) - bupperleft(bb,2)) / (bupperright(bb,1) -
bupperleft(bb,1));
bupperintercept(bb) = bupperleft(bb,2) - (bupperg(bb) * bupperleft(bb,1));
blowerg(bb) = (blowerright(bb,2) - blowerleft(bb,2)) / (blowerright(bb,1) -
blowerleft(bb,1));
blowerintercept(bb) = blowerleft(bb,2) - (blowerg(bb) * blowerleft(bb,1));
bleftg(bb) = (bupperleft(bb,1) - blowerleft(bb,1)) / (bupperleft(bb,2) -
blowerleft(bb,2));
bleftintercept(bb) = bupperleft(bb,1) - (bleftg(bb) * bupperleft(bb,2));
brightg(bb) = (bupperright(bb,1) - blowerright(bb,1)) / (bupperright(bb,2) -
blowerright(bb,2)) ;
brightintercept(bb) = bupperright(bb,1) - (brightg(bb) * bupperright(bb,2));
end
end

%% Plotting
h1 = plot(swarm(:,1), swarm(:,2), 'x','LineWidth',2,'markersize',8);
hold on
axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')
h2=text(0.85*GrSiz,0.95*GrSiz,[int2str(0) ' (' int2str(0/N*100) '%)']);

%% Swarm evolution
for iter = 1 : N %run N evolutions

swarmx(:,iter) = swarm(:,1);
swarmy(:,iter) = swarm(:,2);

for i = 1 : SwarmSize %determine the next move for each agent

% building limits
if indoor == 1 | outdoor ==1
for bb = 1:buildings
    bupperL(bb) = bupperg(bb) * swarm(i,1) + bupperintercept(bb);
    blowerL(bb) = blowerg(bb) * swarm(i,1) + blowerintercept(bb);
    bleftL(bb) = bleftg(bb) * swarm(i,2) + bleftintercept(bb);
    brightL(bb) = brightg(bb) * swarm(i,2) + brightintercept(bb);
end
end

if iter > 1 %analyze neighboring cells visitations
A=zeros(1,8); %assume none of the neighboring cells is visited
for j=1:8
    if indoor == 1 | outdoor ==1

        % Boundaries and buildings

```

```

if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &...
        swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
A(j) = 1; %cell has been visited already
elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
            swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...
            swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
            swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1 |...
            (swarm(i,1)+CelSz*round(cosd((j-1)*45)) >= bleftL &...
            swarm(i,2)+CelSz*round(sind((j-1)*45)) >= blowerL) &...
            (swarm(i,1)+CelSz*round(cosd((j-1)*45)) <= brightL) &...
            swarm(i,2)+CelSz*round(sind((j-1)*45)) <= bupperL);
A(j) = 9; %prohibited area
end

else
% Boundaries
if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &...
        swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
A(j) = 1; % cell has been visited already
elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
            swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...
            swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
            swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1);
A(j) = 9;
end
end

% Collision avoidance
if collisionavoidance == 1;
if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarm(:,1) &
        swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarm(:,2))
A(j) = 9; %set to prohibited area if there is an existing UGV
end
end
end

% Non Holonomic @ 180
if holonomicity == 180;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
A(5) = 9;
A(6) = 9;
A(7) = 9;
end
if (swarm(i,5) == 1 &swarm(i,6) == 0)
A(4) = 9;
A(5) = 9;
A(6) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
A(6) = 9;
A(7) = 9;
A(8) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
A(1) = 9;
A(2) = 9;
A(3) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
A(1) = 9;

```



```

        A(2) = 9;
        A(8) = 9;
    end
    if (swarm(i,5) == 0 & swarm(i,6) == -1)
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == 1)
        A(1) = 9;
        A(7) = 9;
        A(8) = 9;
    end
    if (swarm(i,5) == 1 & swarm(i,6) == -1)
        A(3) = 9;
        A(4) = 9;
        A(5) = 9;
    end
end
end

% Non Holonomic @ 90
if holonomicity == 90;
    if (swarm(i,5) == 1 & swarm(i,6) == 1)
        A(4) = 9;
        A(5) = 9;
        A(6) = 9;
        A(7) = 9;
        A(8) = 9;
    end
    if (swarm(i,5) == 1 & swarm(i,6) == 0)
        A(3) = 9;
        A(4) = 9;
        A(5) = 9;
        A(6) = 9;
        A(7) = 9;
    end
    if (swarm(i,5) == 0 & swarm(i,6) == 1)
        A(5) = 9;
        A(6) = 9;
        A(7) = 9;
        A(8) = 9;
        A(1) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == -1)
        A(1) = 9;
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
        A(8) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == 0)
        A(3) = 9;
        A(1) = 9;
        A(2) = 9;
        A(8) = 9;
        A(7) = 9;
    end
    if (swarm(i,5) == 0 & swarm(i,6) == -1)
        A(1) = 9;
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
    end
end

```

```

    A(5) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(2) = 9;
    A(1) = 9;
    A(7) = 9;
    A(8) = 9;
    A(6) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
end
end

% Non Holonomic @ 270
if holonomicity == 270;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
    A(6) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(5) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(7) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(2) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(1) = 9;
end

end
if (swarm(i,5) == 0 & swarm(i,6) == -1)
    A(3) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(4) = 9;
end
end

if min(A) == 0; %check if there are unvisited cells around
    B = find(A==0); %find not visited cell(s)
    C = B(randi(numel(B))); %randomly pick one of them
    NM = 1;
elseif min(A) == 9; %if all of next block is restricted
    NM = 0; %set velocity to 0
    C = 0;
else
    B = find(A==9); %check prohibited zones
    ind=setdiff(1:8,B); %exclude directions towards prohibited zones
    D = randi(length(ind));
    C = ind(D); %randomly pick any allowed cell
    NM = 1;
end
end

```

```

        swarm(i,5) = NM*CelSz*round(cosd((C-1)*45)); %compute x transition
        swarm(i,6) = NM*CelSz*round(sind((C-1)*45)); %compute y transition
        swarm(i,1) = swarm(i,1) + swarm(i,5); %update x position
        swarm(i,2) = swarm(i,2) + swarm(i,6); %update y position
    end
end

%% Plot swarm evolutions
h1.XData=swarm(:,1);
h1.YData=swarm(:,2);
h2.String=[int2str(iter) ' (' int2str(iter/N*100) '%)'];
pause(0.000001/iter^3)
if iter > 5
for ii=1:SwarmSize
h4=plot(swarmx(ii,(iter-2:iter)),swarmy(ii,(iter-
2:iter)),'-.g','LineWidth',0.2);
end
end
end

%% Show all trajectories

figure
hold on
for ii=1:SwarmSize
Cl=rand(3,1);
plot(swarmx(ii,:),swarmy(ii,:),'-.','color',Cl,'LineWidth',1)
plot(swarmx(ii,end),swarmy(ii,end),'x','color',Cl,'LineWidth',1)

if outdoor == 1,
    for bb = 1:23;
        fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'g')
        end
        for bb = 24:31;
            fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'k')
            end
        else if indoor == 1,
            for bb = 1:12;
                fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'k')
                end
            end
        end

end

hold off
axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')

%% Compute the occupancy matrix
OcM=zeros(GrSiz,GrSiz);
for ix=1:GrSiz
    for iy=1:GrSiz
        for is=1:SwarmSize
            for it=1:N
                if swarmx(is,it) == iy & swarmy(is,it) == ix
                    OcM(ix,iy)=OcM(ix,iy)+1;
                end
            end
        end
    end
end

```

```

end
end
end
end
end

%% Show the occupancy matrix
figure
spy(OcM), set(gca,'YDir','normal'), axis square
figure
imagesc(OcM), set(gca,'YDir','normal'), axis square, colorbar
xlabel('Crossrange cell'), ylabel('Downrange cell')
figure
mesh(OcM)
xlabel('Crossrange cell'), ylabel('Downrange cell')
zlabel('Number of cell visitations')

mOcM=mean(mean(OcM));
Fv=find(~OcM); pFv=numel(Fv)/GrSiz/GrSiz*100;
text(0.6*GrSiz,0.6*GrSiz,2*SwarmSize,['Av. # of visitations ' num2str(mOcM,
3)])
text(0.6*GrSiz,0.6*GrSiz,2*SwarmSize-10,['Unvisited cells ' num2str(pFv, 3)
'%'])

visitdata(Mainloop,1) = mOcM;
visitdata(Mainloop,2) = 100-pFv;

%reset swarmx and swarmy
clear swarmx
clear swarmy
end
end

```

APPENDIX B. SEARCH PHASE WITH ALVC GUIDANCE

```
for SS = 1000;

ml = 1;
for Mainloop = 1 : ml
close all

%% Defining swarm
% Neighboring cells numeration
%   4   3   2
%   5   X   1
%   6   7   8
N=SS; %number of iterations
SwarmSize = 20; %number of agents in a swarm
collisionavoidance = 1; %1 for on, 0 for off
holonomicity = 90; %360 for "off", 90, 180 270 degree for "ON"

% Swarm Starting position
Center = 0;
Cornertopright = 0;
Cornebttleftfttopright = 0;
Cornerallsides = 0;
Row = 0;
Btrightcorner = 0;
%for indoor/outdoor starting configuration
configuration = 1; %1 2 and 3 for outdoor, 4 for indoor

CelSz=1;           % cell size
GrSiz=99;          % grid size
A = zeros(1,8);
swarm=zeros(SwarmSize,9);

%choose map
outdoor = 1; % outdoor map, impossible city
indoor = 0; % indoor floorplan of one building
\

%% Defining buildings
if outdoor == 1,
run('Buildings_Obstacles.m')
else if indoor == 1,
run('indoor_floorplan.m')
end
end

%% Initial conditions

for s = 1:SwarmSize

if configuration == 1;
swarm(s,1) = 1;
swarm(s,2) = 85;
end

if configuration == 2;
if s <= (SwarmSize /2)
swarm(s,1) = 1;
swarm(s,2) = 85;
```

```

else
swarm(s,1) = (GrSiz);
swarm(s,2) = 85;
end
end

if configuration == 3;
if s <= (SwarmSize/3)
swarm(s,1) = (GrSiz);
swarm(s,2) = 85;
else if s > (SwarmSize /3) & s <= 2*(SwarmSize /3)
swarm(s,1) = 1;
swarm(s,2) = 20;
else if s > (2*(SwarmSize /3))
swarm(s,1) = 1;
swarm(s,2) = 85;
end
end
end
end

if Center == 1;
swarm(s,1) = (GrSiz+1)/2;
swarm(s,2) = (GrSiz+1)/2;
end

if Bttrightcorner == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
end

if Cornertopright == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
end

if Cornerallsides == 1;
if s <= (SwarmSize/4)
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
else if s > (SwarmSize /4) & s <= 2*(SwarmSize /4)
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
else if s > (2*(SwarmSize /4)) & s <= 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = (GrSiz);
else if s > 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = 1;
end
end
end
end
end

if Row == 1 ;
swarm(s,1) = round(((GrSiz)/SwarmSize ) * s) ;
swarm(s,2) = 1;
end

if Cornebttlefttopright == 1;
if s <= (SwarmSize /2)
swarm(s,1) = 1;

```

```

    swarm(s,2) = 1;
    else
    swarm(s,1) = (GrSiz);
    swarm(s,2) = (GrSiz);
    end
end

end
swarm(:,5) = 0; %initial x transition
swarm(:,6) = 0; %initial y transition

%% Building block calucations
if indoor == 1 | outdoor ==1
buildings = size(blowerleft,1);

for bb = 1:buildings;
bupperg(bb) = (bupperright(bb,2) - bupperleft(bb,2))/(bupperright(bb,1)-
bupperleft(bb,1));
bupperintercept(bb) = bupperleft(bb,2) - (bupperg(bb) * bupperleft(bb,1));
blowerg(bb) = (blowerright(bb,2) - blowerleft(bb,2))/(blowerright(bb,1)-
blowerleft(bb,1));
blowerintercept(bb) = blowerleft(bb,2) - (blowerg(bb) * blowerleft(bb,1));
bleftg(bb) = (bupperleft(bb,1) - blowerleft(bb,1))/(bupperleft(bb,2)-
blowerleft(bb,2));
bleftintercept(bb) = bupperleft(bb,1) - (bleftg(bb) * bupperleft(bb,2));
brightg(bb) = (bupperright(bb,1) - blowerright(bb,1))/(bupperright(bb,2)-
blowerright(bb,2)) ;
brightintercept(bb) = bupperright(bb,1) - (brightg(bb) * bupperright(bb,2));

    for ux = 1:99;
        for uy = 1:99;
            if ux > (bleftg(bb) * uy + bleftintercept(bb))
                if ux < (brightg(bb) * uy + brightintercept(bb));
                    if uy > (blowerg(bb) * ux + blowerintercept(bb))
                        if uy < (bupperg(bb) * ux + bupperintercept(bb))
                            unvisited2(round(ux)+1,round(uy)+1) = 99;
                        end
                    end
                end
            end
        end
    end
end

end
end

%% Plotting

h1=plot(swarm(:,1), swarm(:,2), 'x','LineWidth',1);
axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')
h2=text(0.8*GrSiz,0.95*GrSiz,[int2str(0) ' (' int2str(0/N*100) '%)']);

%% Swarm evolution
for iter = 1 : N                % run N evolutions

swarmx(:,iter) = swarm(:,1);
swarmy(:,iter) = swarm(:,2);

%% Improved search algo (record all unvisited square coordinates)
clear uvvsquares
[m,n] = size(swarmx);
for iterrow = 1:n

```

```

        for swarmcol = 1:m
unvisited2(swarmx(swarmcol,iterrow)+1,swarmy(swarmcol,iterrow)+1) = 9; %set
those visited to 9
        end
end
unvisited = unvisited2([2:100],[2:100]);

if sum(sum(unvisited(:, :) == 0)) >= 1 %first round
    for x = 1:GrSiz
        for y = 1:GrSiz
            if find(unvisited(x,y) == 0)
                uvsy = y;
                uvsx = x;
            else
                uvsy = 0;
                uvsx = 0;
            end

            uv1(y, :) = uvsy;
            uv2(y, :) = uvsx;
        end
        uvx(:, x) = uv1;
        uvy(:, x) = uv2;
    end

    if sum(sum(unvisited(:, :) == 0)) == 1;
        nn = iter; %record the iteration number when all cell but one is zero
    end

else %Second round (when all the cell has been found, reset and being from
scratch)
    for iterrow = 1:n
        for swarmcol = 1:m
            unvisited2(swarmx(swarmcol,iterrow)+1,swarmy(swarmcol,iterrow)+1) = 0; %set
those visited in first round to 0
        end
    end
    for iterrow = nn:n %start the recording from iter nn
        for swarmcol = 1:m
            unvisited2(swarmx(swarmcol,iterrow)+1,swarmy(swarmcol,iterrow)+1) = 9; %set
those visited to 9
        end
    end

    unvisited = unvisited2([2:100],[2:100]);

        for x = 1:GrSiz
            for y = 1:GrSiz
                if find(unvisited(x,y) == 0)
                    uvsy = y;
                    uvsx = x;
                else
                    uvsy = 0;
                    uvsx = 0;
                end

                uv1(y, :) = uvsy;
                uv2(y, :) = uvsx;
            end
            uvx(:, x) = uv1;
            uvy(:, x) = uv2;
        end
end

```



```

end

uvsquares(:,2) = uvx(uvx~=0) ; %records all unvisited square x and y axis
uvsquares(:,1) = uvy(uvy~=0) ;

for i = 1 : SwarmSize %determine the next move for each agent

% building limits
if indoor == 1 | outdoor ==1
for bb = 1:buildings
    bupperL(bb) = bupperg(bb) * swarm(i,1) + bupperintercept(bb);
    blowerL(bb) = blowerg(bb) * swarm(i,1) + blowerintercept(bb);
    bleftL(bb) = bleftg(bb) * swarm(i,2) + bleftintercept(bb);
    brightL(bb) = brightg(bb) * swarm(i,2) + brightintercept(bb);
end
end

if iter > 1 %analyze neighboring cells visitations
    A=zeros(1,8); %assume none of the neighboring cells is visited
    for j=1:8
        if indoor == 1 || outdoor ==1
            % Boundaries and buildings
            if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &
                swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
                A(j) = 1; %cell has been visited already
            elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...
                swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1 |...
                (swarm(i,1)+CelSz*round(cosd((j-1)*45)) > bleftL &
                swarm(i,2)+CelSz*round(sind((j-1)*45)) > blowerL) &
                (swarm(i,1)+CelSz*round(cosd((j-1)*45)) < brightL) &
                swarm(i,2)+CelSz*round(sind((j-1)*45)) < bupperL);
                A(j) = 9; %prohibited area
            end
        else
            % Boundaries
            if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &
                swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
                A(j) = 1; %cell has been visited already
            elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...
                swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1);
                A(j) = 9; %prohibited area
            end
        end
    end

    % Collision avoidance
    if collisionavoidance == 1;
    if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarm(:,1) &
        swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarm(:,2))
        A(j) = 9; %set to prohibited area if there is an existing UGV
    end
    end

end

end

%% Improved search algo (find angle)
clear distoswarm2
clear distoswarm
distoswarm = (sqrt( ((swarm(i,1) - uvsquares(:,1)).^2) + ((swarm(i,2) -
uvsquares(:,2)).^2) ));

```

```

distoswarm2 = find(distoswarm == min(distoswarm(distoswarm > 0)) );
DD = randi(length(distoswarm2));
uvsquareselect = uvsquares(distoswarm2(DD),:);

y_opp = uvsquareselect(1,2)-swarm(i,2);
x_adj = uvsquareselect(1,1)-swarm(i,1);

uvsquareselectangle = atand(y_opp/x_adj);
uvsquareselectangle2(i,iter) = uvsquareselectangle;

% define quarter of unvisited square
% quarter 2      quarter 1
%      x
% quarter 2      quarter 1

if (y_opp >= 0 && x_adj >= 0) || (y_opp < 0 && x_adj >= 0)
quarter = 1;
else
quarter = 2;
end
randpir = randi(2);
if quarter == 1 %right side
    if uvsquareselectangle <= 90 && uvsquareselectangle >= 67.5
        p1 = 3;
        p2 = [2;4];
        p3 = [1;5];
        p4 = [6;8];
        p5 = 7;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < 67.5 && uvsquareselectangle >= 22.5
            p1 = 2;
            p2 = [1;3];
            p3 = [4;8];
            p4 = [5;7];
            p5 = 6;
            piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
            p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
            elseif uvsquareselectangle < 22.5 && uvsquareselectangle >= -22.5
                p1 = 1;
                p2 = [2;8];
                p3 = [3;7];
                p4 = [4;6];
                p5 = 5;
                piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
                p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                elseif uvsquareselectangle < -22.5 && uvsquareselectangle >= -67.5
                    p1 = 8;
                    p2 = [1;7];
                    p3 = [2;6];
                    p4 = [3;5];
                    p5 = 4;
                    piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
                    p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                    elseif uvsquareselectangle < -67.5 && uvsquareselectangle >= -90
                        p1 = 7;
                        p2 = [6;8];
                        p3 = [1;5];
                        p4 = [2;4];
                        p5 = 3;
                        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
                        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];

```

```

end

elseif quarter == 2 %left side
    if uvsquareselectangle <= 90 && uvsquareselectangle >= 67.5
        p1 = 7;
        p2 = [6;8];
        p3 = [1;5];
        p4 = [2;4];
        p5 = 3;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < 67.5 && uvsquareselectangle >= 22.5
            p1 = 6;
            p2 = [5;7];
            p3 = [4;8];
            p4 = [1;3];
            p5 = 2;
            piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
            p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
            elseif uvsquareselectangle < 22.5 && uvsquareselectangle >= -22.5
                p1 = 5;
                p2 = [4;6];
                p3 = [3;7];
                p4 = [2;8];
                p5 = 1;
                piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
                p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                elseif uvsquareselectangle < -22.5 && uvsquareselectangle >= -67.5
                    p1 = 4;
                    p2 = [3;5];
                    p3 = [2;6];
                    p4 = [1;7];
                    p5 = 8;
                    piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
                    p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                    elseif uvsquareselectangle < -67.5 && uvsquareselectangle >= -90
                        p1 = 3;
                        p2 = [2;4];
                        p3 = [1;5];
                        p4 = [6;8];
                        p5 = 7;
                        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
                        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                    end
                end
            end
        end
    end

```

```

% Non Holonomic @ 180
if holonomicity == 180;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(6) = 9;
    A(7) = 9;
end

```

```

    A(8) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(1) = 9;
    A(2) = 9;
    A(3) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(1) = 9;
    A(2) = 9;
    A(8) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == -1)
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(1) = 9;
    A(7) = 9;
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
end
end
end

% Non Holonomic @ 90
if holonomicity == 90;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
    A(8) = 9;
    A(1) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(1) = 9;
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
    A(8) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(3) = 9;
    A(1) = 9;
end

```

```

        A(2) = 9;
        A(8) = 9;
        A(7) = 9;
    end
    if (swarm(i,5) == 0 & swarm(i,6) == -1)
        A(1) = 9;
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
        A(5) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == 1)
        A(2) = 9;
        A(1) = 9;
        A(7) = 9;
        A(8) = 9;
        A(6) = 9;
    end
    if (swarm(i,5) == 1 & swarm(i,6) == -1)
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
        A(5) = 9;
        A(6) = 9;
    end
end
end

% Non Holonomic @ 270
if holonomicity == 270;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
    A(6) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(5) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(7) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(2) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(1) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == -1)
    A(3) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(4) = 9;
end
end
end

if min(A) == 0; %check if there are unvisited cells around
    B = find(A==0); %find not visited cell(s)
    C = B(randi(numel(B))); %randomly pick one of them
    NM = 1;
elseif min(A) == 9; %if all of next block is restricted
    NM = 0; %set velocity to 0
end
end

```

```

        C = 0;
    else
        B = find(A==9); %check prohibited zones
        ind=setdiff(1:8,B); %exclude directions towards prohibited zones
        for indi = 1:length(ind)
            C1(1,indi) = find(ind(1,indi) == pirioritycell);
        end
        [temp2 C2] = min(C1);
        C = ind(C2);
        NM = 1;
    end
    swarm(i,5) = NM*CelSz*round(cosd((C-1)*45)); %compute x transition
    swarm(i,6) = NM*CelSz*round(sind((C-1)*45)); %compute y transition
    swarm(i,1) = swarm(i,1) + swarm(i,5); %update x position
    swarm(i,2) = swarm(i,2) + swarm(i,6); %update y position
    clear C1
end
swarmx(:,iter) = swarm(:,1);
swarmy(:,iter) = swarm(:,2);
dirswarmx(:,iter) = swarm(:,5);
dirswarmy(:,iter) = swarm(:,6);
end
%% Plot swarm evolutions

h1.XData=swarm(:,1);
h1.YData=swarm(:,2);
h2.String=[int2str(iter) ' (' int2str(iter/N*100) '%)'];
pause(0.000001/iter^3)
%% Add a couple of trajectories

hold
L=10;
for ii=1:L
    LL=randi(SwarmSize);
    C1=rand(3,1);
    plot(swarmx(LL,:),swarmy(LL,:), '-.', 'color',C1)
    plot(swarmx(LL,end),swarmy(LL,end), 'x', 'color',C1, 'LineWidth',2)
end

%% Show all trajectories

figure
hold on
for ii=1:SwarmSize
    C1=rand(3,1);
    plot(swarmx(ii,:),swarmy(ii,:), '-.', 'color',C1, 'LineWidth',1)
    plot(swarmx(ii,end),swarmy(ii,end), 'x', 'color',C1, 'LineWidth',1)
end

if outdoor == 1,
    for bb = 1:23;
        fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
            bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
            bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'g')
        end
        for bb = 24:31;
            fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
                bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
                bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'k')
            end
    else if indoor == 1,
        for bb = 1:12;

```

```

        fill([dblowerleft(bb,1) dblowerright(bb,1) dbupperright(bb,1)
            dbupperleft(bb,1) dblowerleft(bb,1)], [dblowerleft(bb,2)
            dblowerright(bb,2) dbupperright(bb,2) dbupperleft(bb,2)
            dblowerleft(bb,2)], 'k')
    end
end
end

end
hold off
axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')

%% Compute the occupancy matrix
OcM=zeros(GrSiz,GrSiz);
for ix=1:GrSiz
    for iy=1:GrSiz
        for is=1:SwarmSize
            for it=1:N
                if swarmx(is,it) == iy & swarmy(is,it) == ix
                    OcM(ix,iy)=OcM(ix,iy)+1;
                end
            end
        end
    end
end
end

%% Show the occupancy matrix

figure
spy(OcM), set(gca,'YDir','normal'), axis square
figure
imagesc(OcM), set(gca,'YDir','normal'), axis square, colorbar
xlabel('Crossrange cell'), ylabel('Downrange cell')
figure
mesh(OcM)
xlabel('Crossrange cell'), ylabel('Downrange cell')
zlabel('Number of cell visitations')
mOcM=mean(mean(OcM));
Fv=find(~OcM); pFv=numel(Fv)/GrSiz/GrSiz*100;
visitdata(Mainloop,1) = mOcM;
visitdata(Mainloop,2) = 100-pFv;

%reset swarmx and swarmy
clear swarmx
clear swarmy
end
end

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. TRACK AND ENGAGE PHASE WITH LVC GUIDANCE

```
for SS = 1000; %iteration

ml = 30; %number of runs
for Mainloop = 1 : ml
close all

%% Defining initial conditions
% Neighboring cells numeration
%   4   3   2
%   5   X   1
%   6   7   8

N=SS;           %number of iterations
SwarmSize = 20; %number of agents in swarm
enemies = 5;    %number of enemies
sensor = 15;    %ability for UGV to detect enemy
killdis = 1;    %how far UGV can shoot
killdise = 2;   %how far enemy can shoot
collisionavoidance = 1; %1 for on, 0 for off
holonomicityint = 90; %360 for "off", 90, 180 270 degree for "ON"
holonomicityduringtrack = 90;
PSO = 1; %"1 for on, 0 for off"
pkillswarm = 0.1; %prob that enemy will kill UGV
pkillenemy = 0.1; %prob that UGV will kill enemy
shootsequence = 0; %1 for red shoot first(baseline) / 0 for blue shoot first

%choose map
outdoor = 0; %outdoor map, impossible city
indoor = 0; %indoor floorplan of one building

% Swarm Starting position
Center = 0;
Cornertoprightright = 0;
Cornebtlefttopright = 0;
Cornerallsides = 0;
Row = 0;
Btrightcorner = 0;

% for indoor and outdoor starting configuration
configuration = 1; %1, 2, 3 for outdoor and 4 for indoor

inertia = 1;
correction_factor = 2;

CelSz = 1; %cell size
GrSiz = 99; %grid size
A = zeros(1,8);
swarm = zeros(SwarmSize,9);
swarm(:,5) = 0; %initial x transition
swarm(:,6) = 0; %initial y transition
RM = 0;

%% Defining enemies starting positions
enemy = zeros(enemies,6);
enemy(1, 1) = 90; %starting x
enemy(1, 2) = 80; %starting y
```

```

enemy(2, 1) = 73; %starting x
enemy(2, 2) = 20; %starting y

enemy(3, 1) = 55; %starting x
enemy(3, 2) = 98; %starting y

enemy(4, 1) = 25; %starting x
enemy(4, 2) = 40; %starting y

enemy(5, 1) = 45; %starting x
enemy(5, 2) = 60; %starting y

%% Load map
if outdoor == 1,
run('Buildings_Obstacles.m')
else if indoor == 1,
run('indoor_floorplan.m')
end
end

%% Swarm starting positions code

for s = 1:SwarmSize

    if configuration == 1;
    swarm(s,1) = 1;
    swarm(s,2) = 85;
    end

    if configuration == 2;
    if s <= (SwarmSize /2)
    swarm(s,1) = 1;
    swarm(s,2) = 85;
    else
    swarm(s,1) = (GrSiz);
    swarm(s,2) = 85;
    end
    end

    if configuration == 3;
    if s <= (SwarmSize/3)
    swarm(s,1) = (GrSiz);
    swarm(s,2) = 85;
    else if s > (SwarmSize /3) & s <= 2*(SwarmSize /3)
    swarm(s,1) = 1;
    swarm(s,2) = 20;
    else if s > (2*(SwarmSize /3))
    swarm(s,1) = 1;
    swarm(s,2) = 85;
    end
    end
    end

    if configuration == 4;
    swarm(s,1) = 1;
    swarm(s,2) = 10;
    end

    if Center == 1;
    swarm(s,1) = (GrSiz+1)/2;

```

```

swarm(s,2) = (GrSiz+1)/2;
end

if Bttrightcorner == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
end

if Cornertopright == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
end

if Cornerallsides == 1;
if s <= (SwarmSize/4)
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
else if s > (SwarmSize /4) & s <= 2*(SwarmSize /4)
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
else if s > (2*(SwarmSize /4)) & s <= 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = (GrSiz);
else if s > 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = 1;
end
end
end
end
end
if Row == 1 ;
swarm(s,1) = round(((GrSiz)/SwarmSize ) * s) ;
swarm(s,2) = 1;
end

if Cornebtlefttopright == 1;
if s <= (SwarmSize /2)
swarm(s,1) = 1;
swarm(s,2) = 1;
else
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
end
end
end

end

%% Building block calucations
if indoor == 1 | outdoor ==1
buildings = size(blowerleft,1);

for bb = 1:buildings;
bupperg(bb) = (bupperright(bb,2) - bupperleft(bb,2)) / (bupperright(bb,1) -
bupperleft(bb,1));
bupperintercept(bb) = bupperleft(bb,2) - (bupperg(bb) * bupperleft(bb,1));
blowerg(bb) = (blowerright(bb,2) - blowerleft(bb,2)) / (blowerright(bb,1) -
blowerleft(bb,1));
blowerintercept(bb) = blowerleft(bb,2) - (blowerg(bb) * blowerleft(bb,1));
bleftg(bb) = (bupperleft(bb,1) - blowerleft(bb,1)) / (bupperleft(bb,2) -
blowerleft(bb,2));
bleftintercept(bb) = bupperleft(bb,1) - (bleftg(bb) * bupperleft(bb,2));

```

```

brightg(bb) = (bupperright(bb,1) - blowerright(bb,1))/(bupperright(bb,2)-
blowerright(bb,2)) ;
brightintercept(bb) = bupperright(bb,1) - (brightg(bb) * bupperright(bb,2));
end
end

%% Plotting

h1 = plot(swarm(:,1), swarm(:,2), 'x','LineWidth',1);
hold on
h2 = plot(enemy(:,1), enemy(:,2), 'xr','LineWidth',1);
hold on

if outdoor == 1,

    for bb = 1:23;
fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1) bupperleft(bb,1)
blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2) bupperright(bb,2)
bupperleft(bb,2) blowerleft(bb,2)], 'k')
end

    for bb = 24:31;
fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1) bupperleft(bb,1)
blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2) bupperright(bb,2)
bupperleft(bb,2) blowerleft(bb,2)], 'g')
end

else if indoor == 1,
    for bb = 1:12;
fill([dblowerleft(bb,1) dblowerright(bb,1) dbupperright(bb,1) dbupperleft(bb,1)
dblowerleft(bb,1)], [dblowerleft(bb,2) dblowerright(bb,2) dbupperright(bb,2)
dbupperleft(bb,2) dblowerleft(bb,2)], 'k')
end
end
end

axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')
h3=text(0.85*GrSiz,0.95*GrSiz,[int2str(0) ' (' int2str(0/N*100) '%)']);

%% Swarm evolution
for iter = 1 : N % run N evolutions

swarmx(:,iter) = swarm(:,1);
swarmy(:,iter) = swarm(:,2);

% Enemy movement
for e = 1 : enemies; % position of Swarms

    % enemies space boundaries and building limits
if indoor == 1 | outdoor ==1
for bb = 1:buildings
    bupperLLL(bb) = bupperg(bb) * enemy(e,1) + bupperintercept(bb);
    blowerLLL(bb) = blowerg(bb) * enemy(e,1) + blowerintercept(bb);
    bleftLLL(bb) = bleftg(bb) * enemy(e,2) + bleftintercept(bb);
    brightLLL(bb) = brightg(bb) * enemy(e,2) + brightintercept(bb);
end
end

if iter > 1 % analyze neighboring cells visitations
AA=zeros(1,8); % assume none of the neighboring cells is visited

```

```

for jj=1:8
    if indoor == 1 | outdoor ==1
        if find(enemy(e,1)+CelSz*round(cosd((jj-1)*45)) > GrSiz |...
            enemy(e,2)+CelSz*round(sind((jj-1)*45)) > GrSiz |...
            enemy(e,1)+CelSz*round(cosd((jj-1)*45)) < 1 |...
            enemy(e,2)+CelSz*round(sind((jj-1)*45)) < 1 |...
            (enemy(e,1)+CelSz*round(cosd((jj-1)*45)) > bleftLLL &...
            enemy(e,2)+CelSz*round(sind((jj-1)*45)) > blowerLLL) &...
            (enemy(e,1)+CelSz*round(cosd((jj-1)*45)) < brightLLL) &...
            enemy(e,2)+CelSz*round(sind((jj-1)*45)) < bupperLLL);

            AA(jj) = 9; % prohibited area
        end

        elseif find(enemy(e,1)+CelSz*round(cosd((jj-1)*45)) > GrSiz |...
            enemy(e,2)+CelSz*round(sind((jj-1)*45)) > GrSiz |...
            enemy(e,1)+CelSz*round(cosd((jj-1)*45)) < 1 |...
            enemy(e,2)+CelSz*round(sind((jj-1)*45)) < 1);

            AA(jj) = 9;
        end
    end

    if enemy(e,1) == -54321; % dead position
        enemy(e,5) = 0;
        enemy(e,6) = 0;
    elseif min(AA) == 9;
        NMM = 1;
        CC = randi([1 8]);
    else
        BB = find(AA==9); % check prohibited zones
        ind=setdiff(1:8,BB); % exclude directions towards prohibited zones
        DD = randi(length(ind));
        CC = ind(DD); % randomly pick any allowed cell
        NMM = 1;
        enemy(e,5) = NMM*CelSz*round(cosd((CC-1)*45));%compute x transition
        enemy(e,6) = NMM*CelSz*round(sind((CC-1)*45));%compute y transition
    end

    enemy(e,1) = enemy(e,1) + enemy(e,5); %update x position
    enemy(e,2) = enemy(e,2) + enemy(e,6); %update y position
end

end

%Swarm movement
for i = 1 : SwarmSize % determine the next move for each agent

% building limits
if indoor == 1 | outdoor ==1
for bb = 1:buildings
    bupperL(bb) = bupperg(bb) * swarm(i,1) + bupperintercept(bb);
    blowerL(bb) = blowerg(bb) * swarm(i,1) + blowerintercept(bb);
    bleftL(bb) = bleftg(bb) * swarm(i,2) + bleftintercept(bb);
    brightL(bb) = brightg(bb) * swarm(i,2) + brightintercept(bb);

end
end

if iter > 1 % analyze neighboring cells visitations
    A=zeros(1,8); % assume none of the neighboring cells is visited
    for j=1:8
        if indoor == 1 | outdoor ==1

```

```

% Boundaries and buildings
if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
A(j) = 1; % cell has been visited already
elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...
    swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1 |...
    (swarm(i,1)+CelSz*round(cosd((j-1)*45)) > bleftL &...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) > blowerL) &...
    (swarm(i,1)+CelSz*round(cosd((j-1)*45)) < brightL) &...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) < bupperL);
A(j) = 9; % prohibited area

end

else
% Boundaries
if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
A(j) = 1; % cell has been visited already
elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...
    swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
    swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1);
A(j) = 9;

end
end

% Collision avoidance
if collisionavoidance == 1;
if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarm(:, 1)
&... %find other UGV in surrounding
    swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarm(:, 2))
A(j) = 9; %set to prohibited area if there is an existing UGV
end
end

end

if RM == 1 %holo 360 during track
holonomicity = holonomicityduringtrack;
else
holonomicity = holonomicityint;
end

% Non Holonomic @ 180
if holonomicity == 180;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
A(5) = 9;
A(6) = 9;
A(7) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
A(4) = 9;
A(5) = 9;
A(6) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
A(6) = 9;
A(7) = 9;
A(8) = 9;
end
end

```

```

if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(1) = 9;
    A(2) = 9;
    A(3) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(1) = 9;
    A(2) = 9;
    A(8) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == -1)
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(1) = 9;
    A(7) = 9;
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
end
end
end

% Non Holonomic @ 90
if holonomicity == 90;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
    A(8) = 9;
    A(1) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(1) = 9;
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
    A(8) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(3) = 9;
    A(1) = 9;
    A(2) = 9;
    A(8) = 9;
end

```

```

        A(7) = 9;
    end
    if (swarm(i,5) == 0 & swarm(i,6) == -1)
        A(1) = 9;
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
        A(5) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == 1)
        A(2) = 9;
        A(1) = 9;
        A(7) = 9;
        A(8) = 9;
        A(6) = 9;
    end
    if (swarm(i,5) == 1 & swarm(i,6) == -1)
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
        A(5) = 9;
        A(6) = 9;
    end
end
end

% Non Holonomic @ 270
if holonomicity == 270;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
    A(6) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(5) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(7) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(2) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(1) = 9;
end

end
if (swarm(i,5) == 0 & swarm(i,6) == -1)
    A(3) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(4) = 9;
end
end
end

if PSO == 0 %no PSO, use LVS
    if min(A) == 0; %check if there are unvisited cells around
        B = find(A==0); %find not visited cell(s)
        C = B(randi(numel(B))); %randomly pick one of them
        NM = 1;
    elseif min(A) == 9; %if all of next block is restricted
        NM = 0; %set velocity to 0
    end
end
end

```



```

        C = 0;
    else
        B = find(A==9); %check prohibited zones
        ind=setdiff(1:8,B);%exclude directions towards prohibited zones
        D = randi(length(ind));
        C = ind(D); %randomly pick any allowed cell
        NM = 1;
    end
    else %PSO function is on
        if RM == 1;
            % x vel vector
            priorityx = inertia*swarm(i, 5) + correction_factor*rand*(swarm(i, 3) -
            swarm(i, 1)) + correction_factor*rand*(swarm(swarm(i,9), 1) - swarm(i, 1));
            % y vel vector
            priorityy = inertia*swarm(i, 6) + correction_factor*rand*(swarm(i, 4)...
            - swarm(i, 2)) + correction_factor*rand*(swarm(swarm(i,9), 2) - swarm(i, 2));

            % Maximum distance swarm able to move per time step
            if priorityx >= 0.5;
                priorityx = 1;
            else if priorityx <= -0.5;
                priorityx = -1;
            else
                priorityx = 0;
            end
            end

            if priorityy >= 0.5;
                priorityy = 1;
            else if priorityy <= -0.5;
                priorityy = -1;
            else
                priorityy = 0;
            end
            end

            randpir = randi(2);
            if priorityx == 0 && priorityy == 1
                p1 = 3;
                p2 = [2;4];
                p3 = [1;5];
                p4 = [6;8];
                p5 = 7;
            piroritycell = [p1; p2(randpir); p2(find(p2~=p2(randpir))) ; p3(randpir) ;
            p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                elseif priorityx == 1 && priorityy == 1
                    p1 = 2;
                    p2 = [1;3];
                    p3 = [4;8];
                    p4 = [5;7];
                    p5 = 6;
            piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
            p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                elseif priorityx == 1 && priorityy == 0
                    p1 = 1;
                    p2 = [2;8];
                    p3 = [3;7];
                    p4 = [4;6];
                    p5 = 5;
            piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
            p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
                elseif priorityx == 1 && priorityy == -1

```

```

        p1 = 8;
        p2 = [1;7];
        p3 = [2;6];
        p4 = [3;5];
        p5 = 4;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif priorityx == 0 && priorityy == -1
            p1 = 7;
            p2 = [6;8];
            p3 = [1;5];
            p4 = [2;4];
            p5 = 3;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif priorityx == -1 && priorityy == -1
            p1 = 6;
            p2 = [5;7];
            p3 = [4;8];
            p4 = [1;3];
            p5 = 2;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif priorityx == -1 && priorityy == 0
            p1 = 5;
            p2 = [4;6];
            p3 = [3;7];
            p4 = [2;8];
            p5 = 1;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif priorityx == -1 && priorityy == 1
            p1 = 4;
            p2 = [3;5];
            p3 = [2;6];
            p4 = [1;7];
            p5 = 8;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        end

        if min(A) == 9; %if all of next block is restricted
            NM = 0; %set velocity to 0
            C = 0;
        else
            B = find(A==9); %check prohibited zones
            ind = setdiff(1:8,B); %exclude directions towards
        prohibited zones
            for indi = 1:length(ind)
                C1(1,indi) = find(ind(1,indi) == piroritycell);
            end
            [temp2 C2] = min(C1);
            C = ind(C2);
            NM = 1;
        end

    else
        if min(A) == 0; %check if there are unvisited cells around
            B = find(A==0); %find not visited cell(s)
            C = B(randi(numel(B))); %randomly pick one of them
            NM = 1;
        end
    end
end

```

```

elseif min(A) == 9; %if all of next block is either
occupied or in prohibited zone
    NM = 0; %set velocity to 0
    C = 0;
    else
    B = find(A==9); %check prohibited zones
    ind=setdiff(1:8,B); %exclude directions towards
prohibited zones
    D = randi(length(ind));
    C = ind(D); %randomly pick any allowed cell
    NM = 1;
    end
end
end

if swarm(i,1) == NaN
    swarm(i,5) = 0;
    swarm(i,6) = 0;
else
    swarm(i,5) = NM*CelSz*round(cosd((C-1)*45)); %compute x transition
    swarm(i,6) = NM*CelSz*round(sind((C-1)*45)); %compute y transition
    swarm(i,1) = swarm(i,1) + swarm(i,5); %update x position
    swarm(i,2) = swarm(i,2) + swarm(i,6); %update y position
    clear C1
end
end
% Finding enemy
if find((swarm(i,1) >= enemy(:, 1) - sensor & swarm(i,1) <= enemy(:, 1)+
sensor) & (swarm(i,2) >= enemy(:, 2) - sensor & swarm(i,2) <= enemy(:, 2)+
sensor));
    %to solve if sensor found 2 target
    G = find((swarm(i,1) >= (enemy(:, 1) - sensor) & swarm(i,1) <=
(enemy(:, 1)+ sensor)) & (swarm(i,2) >= (enemy(:, 2) - sensor) & swarm(i,2) <=
(enemy(:, 2)+ sensor))); %find and record which enemy is found

    G2 = sqrt((enemy(G,1)-swarm(i,1)).^2 + (enemy(G,2)-
swarm(i,2)).^2); %calculate distance from enemy found
    [temp, G3] = min(G2); %take shorter distance
    enemyfound(i,1) = G(G3,1); %record enemynumber as enemyfound
    enemytarget(i,1) = G(G3,1);
else
    enemyfound(i,1) = 0;
end

if enemyfound(i,1) > 0;
    closestEposu(i,1) = enemy(enemyfound(i,1),1);
    closestEposv(i,1) = enemy(enemyfound(i,1),2);
else
    closestEposu(i,1) = 0;
    closestEposv(i,1) = 0;
end
end
%% Allocating closest ally's PBEST and GBEST that manage to find enemy
if PSO == 1
    if max(enemyfound(:,1)) > 0
        x = find(enemyfound > 0); %other swarm found enemy

        for i = 1:SwarmSize;
            if enemyfound(i,1) == 0;

```

```

        [temp G5] = min(sqrt(((swarm(i,1)-swarm(x,1)).^2) +
        ((swarm(i,2)-swarm(x,2)).^2))); % finding which enemy closest ally (min hypo)
found
        G4 = x(G5,1);
        closestEposu(i,1) = closestEposu(G4,1); %allocate enemy
target position to swarm that did not find enemy
        closestEposv(i,1) = closestEposv(G4,1); %allocate enemy
target position to swarm that did not find enemy
        enemytarget(i,1) = enemyfound(G4,1); %record which enemy
targeted by swarm that did not find enemy
    end
    swarm(i,7) = sqrt(((swarm(i,1)-closestEposu(i,1)).^2) +
    ((swarm(i,2)-closestEposv(i,1)).^2)); %dis from all swarm
position to target enemy position
    end
end

%% Pbest
% comparing previous and current position relative to pbest
    for i = 1:SwarmSize;
        if iter > 1; %only after 1st iter we will have old swarm
position to compare
            valueO(i,1) = sqrt(((swarmx(i,iter-1) - closestEposu(i,1))^2) +
            ((swarmy(i,iter-1) - closestEposv(i,1))^2));%old position wrt new
enemy targeted pos
            valueN(i,1) = sqrt(((swarm(i,1) - closestEposu(i,1))^2) +
            ((swarm(i,2) - closestEposv(i,1))^2)); %new position wrt new enemy
targeted pos

            if valueN(i,1) < valueO(i,1); %if new position is better,
record it as pbest
                swarm(i, 3) = swarm(i, 1); %update best position of u,
                swarm(i, 4) = swarm(i, 2); %update best positions of v,
            else
                swarm(i,3) = swarmx(i,iter-1);
                swarm(i,4) = swarmy(i,iter-1);
            end
        end
    end

if max(enemyfound(:,1)) > 0; %trigger random walk or PSO
    RM = 1; %trigger PSO
% Group similar targeted enemy and assign gbest to min value in each group
    uv = unique(enemytarget); %remove duplicate
    B = size(uv,1);

    for Q = 1:B; %sorting on target found
        A = find(enemytarget(:,1) == uv(Q,1));
        swarm(A,8) = min(swarm(A,7)) ; %min value is Gbest value
    end
    for i = 1:SwarmSize;
        choosegbest = (find(swarm(i,8) == swarm(:,7)));
        randomIndex = randi(length(choosegbest),1);
        swarm(i,9) = choosegbest(randomIndex); %finding which swarm
holds Gbest value recording in 9
        clear choosegbest
        clear randomIndex
    end
else
    RM = 0; % trigger random walk if no enemy found at all
end
end

```

```

end
%% Engage
for i = 1 : SwarmSize;
if shootsequence == 1 %red to shoot first
    if find((swarm(i,1) >= enemy(:, 1) - killdise & swarm(i,1) <= enemy(:, 1)+
        killdise) &...
        (swarm(i,2) >= enemy(:, 2) - killdise & swarm(i,2) <= enemy(:, 2)+
        killdise));
        if rand <= pkillswarm;
            swarm(i,1) = NaN;
            swarm(i,2) = NaN;
        end
    end

    if find((swarm(i,1) >= enemy(:, 1) - killdis & swarm(i,1) <= enemy(:, 1)+
        killdis) & (swarm(i,2) >= enemy(:, 2) - killdis & swarm(i,2) <= enemy(:, 2)+
        killdis));

        K = find((swarm(i,1) >= (enemy(:, 1) - killdis) & swarm(i,1) <= (enemy(:,
        1)+ killdis)) & (swarm(i,2) >= (enemy(:, 2) - killdis) & swarm(i,2) <=
        (enemy(:, 2)+ killdis))); %find and record which enemy is found

        if rand <= pkillenemy;
            enemy(K,1) = -54321;
            enemy(K,2) = -54321;
        end
    end
else %blue to shoot first (reverse order)
    if find((swarm(i,1) >= enemy(:, 1) - killdis & swarm(i,1) <= enemy(:, 1)+
        killdis) & (swarm(i,2) >= enemy(:, 2) - killdis & swarm(i,2) <= enemy(:, 2)+
        killdis));

        K = find((swarm(i,1) >= (enemy(:, 1) - killdis) & swarm(i,1) <= (enemy(:,
        1)+ killdis)) & (swarm(i,2) >= (enemy(:, 2) - killdis) & swarm(i,2) <=
        (enemy(:, 2)+ killdis))); %find and record which enemy is found

        if rand <= pkillenemy;
            enemy(K,1) = -54321;
            enemy(K,2) = -54321;
        end
    end

    if find((swarm(i,1) >= enemy(:, 1) - killdise & swarm(i,1) <= enemy(:, 1)+
        killdise) & (swarm(i,2) >= enemy(:, 2) - killdise & swarm(i,2) <= enemy(:,
        2)+ killdise));

        if rand <= pkillswarm;
            swarm(i,1) = NaN;
            swarm(i,2) = NaN;
        end
    end
end
end

%% Plot swarm evolutions

h1.XData=swarm(:,1);
h1.YData=swarm(:,2);
h2.XData=enemy(:,1);
h2.YData=enemy(:,2);
h3.String=[int2str(iter) ' (' int2str(iter/N*100) '%)'];

```

```

pause(0.000001/iter^3)

% break if all swarm or enemy killed
outcome(Mainloop,2) = sum(enemy(:,1)>=0);
outcome(Mainloop,3) = iter;
outcome(Mainloop,1) = sum(swarm(:,1)>=0); %blue left

if max(~isnan(swarm(:,1))) == 0
break
end

if (sum(enemy(:,1)>=0)) == 0
break
end

end

figure
hold on
for ii=1:SwarmSize
Cl=rand(3,1);
plot(swarmx(ii,:),swarmy(ii,:), '-.', 'color',Cl, 'LineWidth',1)
plot(swarmx(ii,end),swarmy(ii,end), 'x', 'color',Cl, 'LineWidth',1)

if outdoor == 1,
    for bb = 1:23;
        fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'k')
        end
        for bb = 24:31;
            fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'g')
            end
    else if indoor == 1,
        for bb = 1:12;
            fill([dblowerleft(bb,1) dblowerright(bb,1) dbupperright(bb,1)
dbupperleft(bb,1) dblowerleft(bb,1)], [dblowerleft(bb,2)
dblowlerright(bb,2) dbupperright(bb,2) dbupperleft(bb,2)
dblowlerright(bb,2)], 'k')
            end
        end
    end
end

end
hold off
axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')

%% Compute the occupancy matrix
Ocm=zeros(GrSiz,GrSiz);
for ix=1:GrSiz
    for iy=1:GrSiz
        for is=1:SwarmSize
            for it=1:iter
                if swarmx(is,it) == iy & swarmy(is,it) == ix
                    Ocm(ix,iy)=Ocm(ix,iy)+1;
                end
            end
        end
    end
end
end

```

```

end

%% Show the occupancy matrix

figure
spy(OcM), set(gca,'YDir','normal'), axis square
figure
imagesc(OcM), set(gca,'YDir','normal'), axis square, colorbar
xlabel('Crossrange cell'), ylabel('Downrange cell')
figure
mesh(OcM)
xlabel('Crossrange cell'), ylabel('Downrange cell')
zlabel('Number of cell visitations')
mOcM=mean(mean(OcM));
% find percentage in terms of available cells(removing buildings)
if outdoor == 0 && indoor == 0
Fv=find(~OcM); pFv=numel(Fv)/GrSiz/GrSiz*100;
else if outdoor == 1
Fv=find(~OcM); pFv=((numel(Fv)/GrSiz/GrSiz)*GrSiz^2-4674)/((GrSiz^2)-
4674))*100;
    else if indoor == 1
Fv=find(~OcM); pFv=((numel(Fv)/GrSiz/GrSiz)*GrSiz^2-768)/((GrSiz^2)-
768))*100;

% (( total unvisited * totalgridarea ) - building area)/available area *
100%
%building area of 4674 is known by running full coverage and finding out the
max
%amt of percentage that the UGV can cover
    end
end
end

clear swarmx
clear swarmy

end
end

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. TRACK AND ENGAGE PHASE WITH ALVC GUIDANCE

```
for SS = 1000; %iteration

ml = 30; %number of runs
for Mainloop = 1 : ml
close all

%% Defining initial conditions
% Neighboring cells numeration
%   4   3   2
%   5   X   1
%   6   7   8

N=SS;           %number of iterations
SwarmSize = 20; %number of agents in swarm
enemies = 5;    %number of enemies
sensor = 15;    %ability for UGV to detect enemy
killdis = 1;    %how far UGV can shoot
killdise = 2;   %how far enemy can shoot
collisionavoidance = 1; %1 for on, 0 for off
holonomicityint = 90; %360 for "off", 90, 180 270 degree for "ON"
holonomicityduringtrack = 90;
PSO = 1; %"1 for on, 0 for off"
pkillswarm = 0.1; %prob that enemy will kill UGV
pkillenemy = 0.1; %prob that UGV will kill enemy
shootsequence = 0; %1 for red shoot first(baseline) / 0 for blue shoot first

%choose map
outdoor = 0; %outdoor map, impossible city
indoor = 0; %indoor floorplan of one building

% Swarm Starting position
Center = 0;
Cornertoprightright = 0;
Cornebtlefttopright = 0;
Cornerallsides = 0;
Row = 0;
Btrightcorner = 0;

% for indoor and outdoor starting configuration
configuration = 1; %1, 2, 3 for outdoor and 4 for indoor

inertia = 1;
correction_factor = 2;

CelSz = 1; %cell size
GrSiz = 99; %grid size
A = zeros(1,8);
swarm = zeros(SwarmSize,9);
swarm(:,5) = 0; %initial x transition
swarm(:,6) = 0; %initial y transition
RM = 0;

%% Defining enemies starting positions
enemy=zeros(enemies,6);

enemy(1, 1) = 90; %starting x
```

```

enemy(1, 2) = 80; %starting y

enemy(2, 1) = 73; %starting x
enemy(2, 2) = 20; %starting y

enemy(3, 1) = 55; %starting x
enemy(3, 2) = 98; %starting y

enemy(4, 1) = 25; %starting x
enemy(4, 2) = 40; %starting y

enemy(5, 1) = 45; %starting x
enemy(5, 2) = 60; %starting y

%% Load map
if outdoor == 1,
run('Buildings_Obstacles.m')
else if indoor == 1,
run('indoor_floorplan.m')
end
end

%% Swarm starting positions code

for s = 1:SwarmSize

    if configuration == 1;
    swarm(s,1) = 1;
    swarm(s,2) = 85;
    end

    if configuration == 2;
    if s <= (SwarmSize/2)
    swarm(s,1) = 1;
    swarm(s,2) = 85;
    else
    swarm(s,1) = (GrSiz);
    swarm(s,2) = 85;
    end
    end

    if configuration == 3;
    if s <= (SwarmSize/3)
    swarm(s,1) = (GrSiz);
    swarm(s,2) = 85;
    else if s > (SwarmSize /3) & s <= 2*(SwarmSize /3)
    swarm(s,1) = 1;
    swarm(s,2) = 20;
    else if s > (2*(SwarmSize /3))
    swarm(s,1) = 1;
    swarm(s,2) = 85;
    end
    end
    end
    end

    if configuration == 4;
    swarm(s,1) = 1;
    swarm(s,2) = 10;
    end

    if Center == 1;

```

```

swarm(s,1) = (GrSiz+1)/2;
swarm(s,2) = (GrSiz+1)/2;
end

if Bttrightcorner == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
end

if Cornertoprightright == 1;
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
end

if Cornerallsides == 1;
if s <= (SwarmSize/4)
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
else if s > (SwarmSize /4) & s <= 2*(SwarmSize /4)
swarm(s,1) = (GrSiz);
swarm(s,2) = 1;
else if s > (2*(SwarmSize /4)) & s <= 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = (GrSiz);
else if s > 3*(SwarmSize /4)
swarm(s,1) = 1;
swarm(s,2) = 1;
end
end
end
end
end
if Row == 1 ;
swarm(s,1) = round(((GrSiz)/SwarmSize ) * s) ;
swarm(s,2) = 1;
end

if Cornebtlefttopright == 1;
if s <= (SwarmSize /2)
swarm(s,1) = 1;
swarm(s,2) = 1;
else
swarm(s,1) = (GrSiz);
swarm(s,2) = (GrSiz);
end
end
end

end

%% Building block calucations
if indoor == 1 | outdoor ==1
buildings = size(blowerleft,1);

for bb = 1:buildings;
bupperg(bb) = (bupperright(bb,2) - bupperleft(bb,2)) / (bupperright(bb,1) -
bupperleft(bb,1));
bupperintercept(bb) = bupperleft(bb,2) - (bupperg(bb) * bupperleft(bb,1));
blowerg(bb) = (blowerright(bb,2) - blowerleft(bb,2)) / (blowerright(bb,1) -
blowerleft(bb,1));
blowerintercept(bb) = blowerleft(bb,2) - (blowerg(bb) * blowerleft(bb,1));
bleftg(bb) = (bupperleft(bb,1) - blowerleft(bb,1)) / (bupperleft(bb,2) -
blowerleft(bb,2));
bleftintercept(bb) = bupperleft(bb,1) - (bleftg(bb) * bupperleft(bb,2));

```

```

brightg(bb) = (bupperright(bb,1) - blowerright(bb,1))/(bupperright(bb,2) -
blowerright(bb,2)) ;
brightintercept(bb) = bupperright(bb,1) - (brightg(bb) * bupperright(bb,2));
    for ux = 1:99;
        for uy = 1:99;
            if ux > (bleftg(bb) * uy + bleftintercept(bb))
                if ux < (brightg(bb) * uy + brightintercept(bb));
                    if uy > (blowerg(bb) * ux + blowerintercept(bb))
                        if uy < (bupperg(bb) * ux + bupperintercept(bb))
                            unvisited2(round(ux)+1,round(uy)+1) = 99;
                        end
                    end
                end
            end
        end
    end
end

end
end

%% Plotting

h1 = plot(swarm(:,1), swarm(:,2), 'x','LineWidth',1);
hold on
h2 = plot(enemy(:,1), enemy(:,2), 'xr','LineWidth',1);
hold on

%plot map on figure
if outdoor == 1,
    for bb = 1:23;
        fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'k')
        end
    for bb = 24:31;
        fill([blowerleft(bb,1) blowerright(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerright(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'g')
        end
else if indoor == 1,
    for bb = 1:12;
        fill([dblowerleft(bb,1) dblowerright(bb,1) dbupperright(bb,1)
dbupperleft(bb,1) dblowerleft(bb,1)], [dblowerleft(bb,2)
dblowerright(bb,2) dbupperright(bb,2) dbupperleft(bb,2)
dblowerleft(bb,2)], 'k')
        end
    end
end

axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')
h3=text(0.85*GrSiz,0.95*GrSiz,[int2str(0) ' (' int2str(0/N*100) '%)']);

%% Swarm evolution
for iter = 1 : N %run N evolutions

swarmx(:,iter) = swarm(:,1);
swarmy(:,iter) = swarm(:,2);

```

```

%% Improved search algo (record all unvisited sqare coordinates)
clear uvvsquares
[m,n] = size(swarmx);
swarmxx = swarmx;
swarmyy = swarmy;
swarmxx(isnan(swarmxx)) = 0;
swarmyy(isnan(swarmyy)) = 0;
for iterrow = 1:n
    for swarmcol = 1:m
        unvisited2(swarmxx(swarmcol,iterrow)+1,swarmyy(swarmcol,iterrow)+1) = 9; %set
        those visited to 9
    end
end
unvisited = unvisited2([2:100],[2:100]);

if sum(sum(unvisited(:,) == 0)) >= 1 %first round
    for x = 1:GrSiz
        for y = 1:GrSiz
            if find(unvisited(x,y) == 0)
                uvvy = y;
                uvvx = x;
            else
                uvvy = 0;
                uvvx = 0;
            end

            uv1(y,:) = uvvy;
            uv2(y,:) = uvvx;
        end
        uvx(:,x) = uv1;
        uvy(:,x) = uv2;
    end
    nn = iter+1;

else %Second round (when all the cell has been found, reset and being from
scratch)

    for iterrow = 1:n
        for swarmcol = 1:m
            unvisited2(swarmxx(swarmcol,iterrow)+1,swarmyy(swarmcol,iterrow)+1) =
0; %set those visited in first round to 0
        end
    end
    for iterrow = nn:n %start the recording from iter nn
        for swarmcol = 1:m
            unvisited2(swarmxx(swarmcol,iterrow)+1,swarmyy(swarmcol,iterrow)+1) =
9; %set those visited to 9
        end
    end

    unvisited = unvisited2([2:100],[2:100]);

    if sum(sum(unvisited(:,) == 0)) == 0;
        nn = iter %record the iteration number when all cell is zero
    end
    if sum(sum(unvisited(:,) == 0)) == 1; %loop to reset
        nn = iter+1 %record the iteration number when all cell is zero
    end
    if sum(sum(unvisited(:,) == 0)) == 2; %loop to reset
        nn = iter+1 %record the iteration number when all cell is zero
    end
end

```

```

for x = 1:GrSiz
    for y = 1:GrSiz
        if find(unvisited(x,y) == 0)
            uvsy = y;
            uvsx = x;
        else
            uvsy = 0;
            uvsx = 0;
        end
    end

    uv1(y,:) = uvsy;
    uv2(y,:) = uvsx;
    end
    uvx(:,x) = uv1;
    uvy(:,x) = uv2;
    end
end

uvsquares(:,2) = uvx(uvx~=0) ; %records all unvisited square x and y axis
uvsquares(:,1) = uvy(uvy~=0) ;

% enemy movement
for e = 1 : enemies; % position of Swarms

    % enemies space boundaries
    % building limits
    if indoor == 1 | outdoor ==1
        for bb = 1:buildings
            bupperLLL(bb) = bupperg(bb) * enemy(e,1) + bupperintercept(bb);
            blowerLLL(bb) = blowerg(bb) * enemy(e,1) + blowerintercept(bb);
            bleftLLL(bb) = bleftg(bb) * enemy(e,2) + bleftintercept(bb);
            brightLLL(bb) = brightg(bb) * enemy(e,2) + brightintercept(bb);
        end
    end

    if iter > 1 %analyze neighboring cells visitations
        AA=zeros(1,8); %assume none of the neighboring cells is visited
        for jj=1:8
            if indoor == 1 | outdoor ==1
                if find(enemy(e,1)+CelSz*round(cosd((jj-1)*45)) > GrSiz |...
                    enemy(e,2)+CelSz*round(sind((jj-1)*45)) > GrSiz |...
                    enemy(e,1)+CelSz*round(cosd((jj-1)*45)) < 1 |...
                    enemy(e,2)+CelSz*round(sind((jj-1)*45)) < 1 |...
                    (enemy(e,1)+CelSz*round(cosd((jj-1)*45)) > bleftLLL) &...
                    enemy(e,2)+CelSz*round(sind((jj-1)*45)) > blowerLLL) &...
                    (enemy(e,1)+CelSz*round(cosd((jj-1)*45)) < brightLLL) &...
                    enemy(e,2)+CelSz*round(sind((jj-1)*45)) < bupperLLL);
                    AA(jj) = 9; %prohibited area
                end

                elseif find(enemy(e,1)+CelSz*round(cosd((jj-1)*45)) > GrSiz |...
                    enemy(e,2)+CelSz*round(sind((jj-1)*45)) > GrSiz |...
                    enemy(e,1)+CelSz*round(cosd((jj-1)*45)) < 1 |...
                    enemy(e,2)+CelSz*round(sind((jj-1)*45)) < 1);
                    AA(jj) = 9;
                end
            end
        end
    end

    if enemy(e,1) == -54321; %dead position
        enemy(e,5) = 0;
        enemy(e,6) = 0;
    end
end

```

```

elseif min(AA) == 9;
    NMM = 1;
    CC = randi([1 8]);
    else
        BB = find(AA==9); %check prohibited zones
        ind=setdiff(1:8,BB); %exclude directions towards prohibited
zones
        DD = randi(length(ind));
        CC = ind(DD); %randomly pick any allowed cell
        NMM = 1;

        enemy(e,5) = NMM*CelSz*round(cosd((CC-1)*45));%compute x transition
        enemy(e,6) = NMM*CelSz*round(sind((CC-1)*45));%compute y transition
    end

    enemy(e,1) = enemy(e,1) + enemy(e,5); %update x position
    enemy(e,2) = enemy(e,2) + enemy(e,6); %update y position
end

end

%Swarm movement
for i = 1 : SwarmSize % determine the next move for each agent

% building limits
if indoor == 1 | outdoor ==1
for bb = 1:buildings
    bupperL(bb) = bupperg(bb) * swarm(i,1) + bupperintercept(bb);
    blowerL(bb) = blowerg(bb) * swarm(i,1) + blowerintercept(bb);
    bleftL(bb) = bleftg(bb) * swarm(i,2) + bleftintercept(bb);
    brightL(bb) = brightg(bb) * swarm(i,2) + brightintercept(bb);

end
end

if iter > 1 % analyze neighboring cells visitations
    A = zeros(1,8); % assume none of the neighboring cells is visited
    for j = 1:8
        if indoor == 1 | outdoor ==1

            % Boundaries and buildings
            if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
                A(j) = 1; % cell has been visited already
            elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...
                swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1 |...
                (swarm(i,1)+CelSz*round(cosd((j-1)*45)) > bleftL &...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) > blowerL) &...
                (swarm(i,1)+CelSz*round(cosd((j-1)*45)) < brightL) &...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) < bupperL);
                A(j) = 9; % prohibited area
            end

        else
            % Boundaries
            if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarmx(:, :) &...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarmy(:, :))
                A(j) = 1; % cell has been visited already
            elseif find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) > GrSiz |...
                swarm(i,2)+CelSz*round(sind((j-1)*45)) > GrSiz |...

```

```

        swarm(i,1)+CelSz*round(cosd((j-1)*45)) < 1 |...
        swarm(i,2)+CelSz*round(sind((j-1)*45)) < 1);
    A(j) = 9;
    end
    end

    % Collision avoidance
    if collisionavoidance == 1;
    if find(swarm(i,1)+CelSz*round(cosd((j-1)*45)) == swarm(:,1) &...
        swarm(i,2)+CelSz*round(sind((j-1)*45)) == swarm(:,2))
        A(j) = 9; %set to prohibited area if there is an existing UGV
    end
    end
end

    %% Improved search algo (find angle)
clear distoswarm2
clear distoswarm
if sum(swarm(i,1)) > 0
distoswarm = (sqrt( ((swarm(i,1) - uvsquares(:,1)).^2) + ((swarm(i,2) -
uvsquares(:,2)).^2) ));
distoswarm2 = find(distoswarm == min(distoswarm(distoswarm > 0)) );DD =
randi(length(distoswarm2));
uvsquareselect = uvsquares(distoswarm2(DD),:);

y_opp = uvsquareselect(1,2)-swarm(i,2);
x_adj = uvsquareselect(1,1)-swarm(i,1);

uvsquareselectangle = atand(y_opp/x_adj);
uvsquareselectangle2(i,iter) = uvsquareselectangle;

% define quarter of unvisited square
% quarter 2      quarter 1
%      x
% quarter 2      quarter 1

if (y_opp >= 0 && x_adj >= 0) || (y_opp < 0 && x_adj >= 0)
quarter = 1;
else
quarter = 2;
end
randpir = randi(2);
if quarter == 1 %right side
    if uvsquareselectangle <= 90 && uvsquareselectangle >= 67.5
        p1 = 3;
        p2 = [2;4];
        p3 = [1;5];
        p4 = [6;8];
        p5 = 7;
    piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
    p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < 67.5 && uvsquareselectangle >= 22.5
            p1 = 2;
            p2 = [1;3];
            p3 = [4;8];
            p4 = [5;7];
            p5 = 6;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
            elseif uvsquareselectangle < 22.5 && uvsquareselectangle >= -22.5
                p1 = 1;
                p2 = [2;8];

```



```

        p3 = [3;7];
        p4 = [4;6];
        p5 = 5;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < -22.5 && uvsquareselectangle >= -67.5
            p1 = 8;
            p2 = [1;7];
            p3 = [2;6];
            p4 = [3;5];
            p5 = 4;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < -67.5 && uvsquareselectangle >= -90
            p1 = 7;
            p2 = [6;8];
            p3 = [1;5];
            p4 = [2;4];
            p5 = 3;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        end

elseif quarter == 2 %left side
    if uvsquareselectangle <= 90 && uvsquareselectangle >= 67.5
        p1 = 7;
        p2 = [6;8];
        p3 = [1;5];
        p4 = [2;4];
        p5 = 3;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < 67.5 && uvsquareselectangle >= 22.5
            p1 = 6;
            p2 = [5;7];
            p3 = [4;8];
            p4 = [1;3];
            p5 = 2;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < 22.5 && uvsquareselectangle >= -22.5
            p1 = 5;
            p2 = [4;6];
            p3 = [3;7];
            p4 = [2;8];
            p5 = 1;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < -22.5 && uvsquareselectangle >= -67.5
            p1 = 4;
            p2 = [3;5];
            p3 = [2;6];
            p4 = [1;7];
            p5 = 8;
        piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
        p3(find(p3~=p3(randpir))) ; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif uvsquareselectangle < -67.5 && uvsquareselectangle >= -90
            p1 = 3;
            p2 = [2;4];
            p3 = [1;5];
            p4 = [6;8];
            p5 = 7;

```

```

piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir))); p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
end
end
    if RM == 1
        holonomicity = 360;
    else
        holonomicity = holonomicityint;
    end

    % Non Holonomic @ 180
    if holonomicity == 180;
    if (swarm(i,5) == 1 & swarm(i,6) == 1)
        A(5) = 9;
        A(6) = 9;
        A(7) = 9;
    end
    if (swarm(i,5) == 1 & swarm(i,6) == 0)
        A(4) = 9;
        A(5) = 9;
        A(6) = 9;
    end
    if (swarm(i,5) == 0 & swarm(i,6) == 1)
        A(6) = 9;
        A(7) = 9;
        A(8) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == -1)
        A(1) = 9;
        A(2) = 9;
        A(3) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == 0)
        A(1) = 9;
        A(2) = 9;
        A(8) = 9;
    end
    if (swarm(i,5) == 0 & swarm(i,6) == -1)
        A(2) = 9;
        A(3) = 9;
        A(4) = 9;
    end
    if (swarm(i,5) == -1 & swarm(i,6) == 1)
        A(1) = 9;
        A(7) = 9;
        A(8) = 9;
    end
    if (swarm(i,5) == 1 & swarm(i,6) == -1)
        A(3) = 9;
        A(4) = 9;
        A(5) = 9;
    end
end
end

    % Non Holonomic @ 90
    if holonomicity == 90;
    if (swarm(i,5) == 1 & swarm(i,6) == 1)
        A(4) = 9;
        A(5) = 9;
        A(6) = 9;
        A(7) = 9;
        A(8) = 9;
    end
end
end

```

```

end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(5) = 9;
    A(6) = 9;
    A(7) = 9;
    A(8) = 9;
    A(1) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(1) = 9;
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
    A(8) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(3) = 9;
    A(1) = 9;
    A(2) = 9;
    A(8) = 9;
    A(7) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == -1)
    A(1) = 9;
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(2) = 9;
    A(1) = 9;
    A(7) = 9;
    A(8) = 9;
    A(6) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(2) = 9;
    A(3) = 9;
    A(4) = 9;
    A(5) = 9;
    A(6) = 9;
end
end
end

% Non Holonomic @ 270
if holonomicity == 270;
if (swarm(i,5) == 1 & swarm(i,6) == 1)
    A(6) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == 0)
    A(5) = 9;
end
if (swarm(i,5) == 0 & swarm(i,6) == 1)
    A(7) = 9;
end

```

```

end
if (swarm(i,5) == -1 & swarm(i,6) == -1)
    A(2) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 0)
    A(1) = 9;

end
if (swarm(i,5) == 0 & swarm(i,6) == -1)
    A(3) = 9;
end
if (swarm(i,5) == -1 & swarm(i,6) == 1)
    A(8) = 9;
end
if (swarm(i,5) == 1 & swarm(i,6) == -1)
    A(4) = 9;
end
end

if PSO == 0 %no PSO, use LVS
    if min(A) == 0; %check if there are unvisited cells around
        B = find(A==0); %find not visited cell(s)
        C = B(randi(numel(B))); %randomly pick one of them
        NM = 1;
    elseif min(A) == 9; %if all of next block is either occupied or in
prohibited zone
        NM = 0; %set velocity to 0
        C = 0;
    else
        B = find(A==9); %check prohibited zones
        ind=setdiff(1:8,B); %exclude directions towards prohibited zones
        for indi = 1:length(ind)
            C1(1,indi) = find(ind(1,indi) == piroritycell);
        end
        [temp2 C2] = min(C1);
        C = ind(C2);
        NM = 1;
    end

    else %PSO function is on
        if RM == 1;
            % x vel vector
            priorityx = inertia*swarm(i, 5) + correction_factor*rand*(swarm(i, 3) -
            swarm(i, 1)) + correction_factor*rand*(swarm(swarm(i,9), 1) - swarm(i,
            1))+rand()-1/2 ;
            % y vel vector
            priorityy = inertia*swarm(i, 6) + correction_factor*rand*(swarm(i, 4) -
            swarm(i, 2)) + correction_factor*rand*(swarm(swarm(i,9), 2) - swarm(i,
            2))+rand()-1/2;

            % Maximum distance swarm able to move per time step
            if priorityx >= 0.5;
                priorityx = 1;
            else if priorityx <= -0.5;
                priorityx = -1;
            else
                priorityx = 0;
            end
            end

            if priorityy >= 0.5;
                priorityy = 1;
            end
        end
    end
end

```

```

else if priorityy <= 0.5;
    priorityy = -1;
else
    priorityy = 0;
end
end

randpir = randi(2);
if priorityx == 0 && priorityy == 1
    p1 = 3;
    p2 = [2;4];
    p3 = [1;5];
    p4 = [6;8];
    p5 = 7;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir)))]; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
elseif priorityx == 1 && priorityy == 1
    p1 = 2;
    p2 = [1;3];
    p3 = [4;8];
    p4 = [5;7];
    p5 = 6;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir)))]; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
elseif priorityx == 1 && priorityy == 0
    p1 = 1;
    p2 = [2;8];
    p3 = [3;7];
    p4 = [4;6];
    p5 = 5;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir)))]; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
elseif priorityx == 1 && priorityy == -1
    p1 = 8;
    p2 = [1;7];
    p3 = [2;6];
    p4 = [3;5];
    p5 = 4;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir)))]; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
elseif priorityx == 0 && priorityy == -1
    p1 = 7;
    p2 = [6;8];
    p3 = [1;5];
    p4 = [2;4];
    p5 = 3;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir)))]; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
elseif priorityx == -1 && priorityy == -1
    p1 = 6;
    p2 = [5;7];
    p3 = [4;8];
    p4 = [1;3];
    p5 = 2;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir)))]; p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
elseif priorityx == -1 && priorityy == 0
    p1 = 5;
    p2 = [4;6];
    p3 = [3;7];
    p4 = [2;8];

```

```

        p5 = 1;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir))); p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        elseif priorityx == -1 && priorityy == 1
            p1 = 4;
            p2 = [3;5];
            p3 = [2;6];
            p4 = [1;7];
            p5 = 8;
piroritycell = [p1; p2(randpir) ; p2(find(p2~=p2(randpir))) ; p3(randpir) ;
p3(find(p3~=p3(randpir))); p4(randpir) ; p4(find(p4~=p4(randpir))) ; p5];
        end

        if min(A) == 9; %if all of next block is either occupied or
in prohibited zone
            NM = 0; %set velocity to 0
            C = 0; %doesn't matter
        else
            B = find(A==9); %check prohibited zones
            ind=setdiff(1:8,B); %exclude directions towards
prohibited zones
            for indi = 1:length(ind)
                C1(1,indi) = find(ind(1,indi) == piroritycell);
            end
            [temp2 C2] = min(C1);
            C = ind(C2);
            NM = 1;
        end

        else
            if min(A) == 0; %check if there are unvisited cells around
            B = find(A==0); %find not visited cell(s)
            C = B(randi(numel(B))); %randomly pick one of them
            NM = 1;
            elseif min(A) == 9; %if all of next block is either
occupied or in prohibited zone
                NM = 0; %set velocity to 0
                C = 0; %doesn't matter
            else
                B = find(A==9); %check prohibited zones
                ind=setdiff(1:8,B); %exclude directions towards
prohibited zones
                for indi = 1:length(ind)
                    C1(1,indi) = find(ind(1,indi) == piroritycell);
                end
                [temp2 C2] = min(C1);
                C = ind(C2);
                NM = 1;
            end
        end
    end

    end

    if swarm(i,1) == NaN
        swarm(i,5) = 0;
        swarm(i,6) = 0;
    else
        swarm(i,5) = NM*CelSz*round(cosd((C-1)*45)); %compute x transition
        swarm(i,6) = NM*CelSz*round(sind((C-1)*45)); %compute y transition
        swarm(i,1) = swarm(i,1) + swarm(i,5); %update x position
        swarm(i,2) = swarm(i,2) + swarm(i,6); %update y position
        clear C1
    end
end

```

```

        end
    end
    % Finding enemy
    if find((swarm(i,1) >= enemy(:, 1) - sensor & swarm(i,1) <= enemy(:, 1)+
    sensor) & (swarm(i,2) >= enemy(:, 2) - sensor & swarm(i,2) <= enemy(:, 2)+
    sensor));
        %To solve if sensor found 2 target
        G = find((swarm(i,1) >= (enemy(:, 1) - sensor) & swarm(i,1) <=
    (enemy(:, 1)+ sensor)) & (swarm(i,2) >= (enemy(:, 2) - sensor) & swarm(i,2) <=
    (enemy(:, 2)+ sensor))); %find and record which enemy is found
        G2 = sqrt((enemy(G,1)-swarm(i,1)).^2 + (enemy(G,2)-
    swarm(i,2)).^2); %calculate distance from enemy found
        [temp, G3] = min(G2); %take shorter distance
        enemyfound(i,1) = G(G3,1); %record enemynumber as enemyfound
        enemytarget(i,1) = G(G3,1);
    else
        enemyfound(i,1) = 0;
    end

    if enemyfound(i,1) > 0;
        closestEposu(i,1) = enemy(enemyfound(i,1),1);
        closestEposv(i,1) = enemy(enemyfound(i,1),2);
    else
        closestEposu(i,1) = 0;
        closestEposv(i,1) = 0;
    end
end

%% Allocating closest ally's PBest and GBest that manage to find enemy
if PSO == 1
    if max(enemyfound(:,1)) > 0
        x = find(enemyfound > 0); %ally that found enemy

        for i = 1:SwarmSize;
            if enemyfound(i,1) == 0;
                [temp G5] = min(sqrt(((swarm(i,1)-swarm(x,1)).^2) +
    ((swarm(i,2)-swarm(x,2)).^2))); %G5 - finding which enemy
                closest ally (min hypo) found
                G4 = x(G5,1);
                closestEposu(i,1) = closestEposu(G4,1); % allocate enemy
                target position to swarm that did not find enemy
                closestEposv(i,1) = closestEposv(G4,1); % allocate enemy
                target position to swarm that did not find enemy
                enemytarget(i,1) = enemyfound(G4,1); %record which enemy
                targeted by swarm that did not find enemy
            end
        end

        swarm(i,7) = sqrt(((swarm(i,1)-closestEposu(i,1)).^2) + ((swarm(i,2)-
    closestEposv(i,1)).^2)); %dis from all swarm poisiton to target enemy position
    end
end

%% Pbest
%comparing previous and current position relative to pbest
for i = 1:SwarmSize;
    if iter >1; %only after 1st iter we will have old swarm
        position to compare
    end
end

```

```

valueO(i,1) = sqrt(((swarmx(i,iter-1) - closestEposu(i,1))^2) +
((swarmy(i,iter-1) - closestEposv(i,1))^2));%old position wrt new
enemy targeted pos
valueN(i,1) = sqrt(((swarm(i,1) - closestEposu(i,1))^2) +
((swarm(i,2) - closestEposv(i,1))^2)); %new position wrt new enemy
targeted pos

if valueN(i,1) < valueO(i,1); %if new position is better,
record it as pbest
    swarm(i, 3) = swarm(i, 1); %update best position of u,
    swarm(i, 4) = swarm(i, 2); %update best positions of v,
else
    swarm(i,3) = swarmx(i,iter-1);
    swarm(i,4) = swarmy(i,iter-1);
end
end
end

if max(enemyfound(:,1)) > 0; %trigger random walk or PSO
    RM = 1; %trigger PSO

% Group similar targeted enemy and assign gbest to min value in each group
uv = unique(enemytarget); %remove duplicate
B = size(uv,1);

for Q = 1:B; % sorting on target found
    A = find(enemytarget(:,1) == uv(Q,1));
    swarm(A,8) = min(swarm(A,7)) ; %min value is Gbest value
end

for i = 1:SwarmSize;
    choosegbest = (find(swarm(i,8) == swarm(:,7)));
    randomIndex = randi(length(choosegbest),1);
    swarm(i,9) = choosegbest(randomIndex); % finding which
    swarm holds Gbest value recording in 9
    clear choosegbest
    clear randomIndex
end
else
    RM = 0; % trigger random walk if no enemy found at all
end

end

%% Engage
for i = 1 : SwarmSize;

if find((swarm(i,1) >= enemy(:, 1) - killldise & swarm(i,1) <= enemy(:, 1)+
killldise) & (swarm(i,2) >= enemy(:, 2) - killldise & swarm(i,2) <= enemy(:, 2)+
killldise));

if rand <= pkillswarm;
swarm(i,1) = NaN;
swarm(i,2) = NaN;
end
end

if find((swarm(i,1) >= enemy(:, 1) - killldis & swarm(i,1) <= enemy(:, 1)+
killldis) & (swarm(i,2) >= enemy(:, 2) - killldis & swarm(i,2) <= enemy(:, 2)+
killldis));

```



```

K = find((swarm(i,1) >= (enemy(:, 1) - killdis) & swarm(i,1) <= (enemy(:, 1)+
killdis)) & (swarm(i,2) >= (enemy(:, 2) - killdis) & swarm(i,2) <= (enemy(:,
2)+ killdis))); %find and record which enemy is found

if rand <= pkillenemy;
enemy(K,1) = -54321;
enemy(K,2) = -54321;
end
end
end

%% Plot swarm evolutions

h1.XData=swarm(:,1);
h1.YData=swarm(:,2);
h2.XData=enemy(:,1);
h2.YData=enemy(:,2);
h3.String=[int2str(iter) ' (' int2str(iter/N*100) '%)'];
pause(0.000001/iter^3)

% break if all swarm or enemy killed
outcome(Mainloop,2) = sum(enemy(:,1)>=0);
outcome(Mainloop,3) = iter;
outcome(Mainloop,1) = sum(swarm(:,1)>=0); %blue left

if max(~isnan(swarm(:,1))) == 0
break
end

if (sum(enemy(:,1)>=0)) == 0
break
end
end
outcome

figure
hold on
for ii=1:SwarmSize
Cl=rand(3,1);
plot(swarmx(ii,:),swarmy(ii,:), '-.', 'color',Cl, 'LineWidth',1)
plot(swarmx(ii,end),swarmy(ii,end), 'x', 'color',Cl, 'LineWidth',1)

if outdoor == 1,
    for bb = 1:23;
        fill([blowerleft(bb,1) blowerleft(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerleft(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'k')
        end
        for bb = 24:31;
            fill([blowerleft(bb,1) blowerleft(bb,1) bupperright(bb,1)
bupperleft(bb,1) blowerleft(bb,1)], [blowerleft(bb,2) blowerleft(bb,2)
bupperright(bb,2) bupperleft(bb,2) blowerleft(bb,2)], 'g')
            end
    else if indoor == 1,
        for bb = 1:12;
            fill([dblowerleft(bb,1) dblowerleft(bb,1) dbupperright(bb,1)
dbupperleft(bb,1) dblowerleft(bb,1)], [dblowerleft(bb,2)
dblowerleft(bb,2) dbupperright(bb,2) dbupperleft(bb,2)
dblowerleft(bb,2)], 'k')
            end
        end
end
end

```

```

end

end
hold off
axis([1 GrSiz 1 GrSiz]), axis square, grid minor
xlabel('Crossrange cell'), ylabel('Downrange cell')

%% Compute the occupancy matrix
OcM=zeros(GrSiz,GrSiz);
for ix=1:GrSiz
    for iy=1:GrSiz
        for is=1:SwarmSize
            for it=1:iter
                if swarmx(is,it) == iy & swarmy(is,it) == ix
                    OcM(ix,iy)=OcM(ix,iy)+1;
                end
            end
        end
    end
end
end

%% Show the occupancy matrix

figure
spy(OcM), set(gca,'YDir','normal'), axis square
figure
imagesc(OcM), set(gca,'YDir','normal'), axis square, colorbar
xlabel('Crossrange cell'), ylabel('Downrange cell')
figure
mesh(OcM)
xlabel('Crossrange cell'), ylabel('Downrange cell')
zlabel('Number of cell visitations')

mOcM=mean(mean(OcM));
% find percentage in terms of available cells(removing buildings)
if outdoor == 0 && indoor == 0
Fv=find(~OcM); pFv=numel(Fv)/GrSiz/GrSiz*100;
else if outdoor == 1
Fv=find(~OcM); pFv=((((numel(Fv)/GrSiz/GrSiz)*GrSiz^2)-4674)/((GrSiz^2)-
4674))*100;
else if indoor == 1
Fv=find(~OcM); pFv=((((numel(Fv)/GrSiz/GrSiz)*GrSiz^2)-768)/((GrSiz^2)-
768))*100;
end
end
end
clear swarmx
clear swarmy
end
enc

```

APPENDIX E. COORDINATES FOR OUTDOOR OBSTACLES

		Lower Left	Lower Right	Upper Left	Upper Right
Building 1	x axis	47	57	56	46
	y axis	89	90	97	96
Building 2	x axis	42.5	49.5	49	42
	y axis	82	83	87	86
Building 3	x axis	33	36	47	44
	y axis	67	64	72	75
Building 4	x axis	22	28	28	22
	y axis	70	70	74	74
Building 5	x axis	26	38	37	25
	y axis	75	80	84	79
Building 6	x axis	18	24	31	25
	y axis	58	54	62	66
Building 7	x axis	34	40	46	40
	y axis	47	43	52	56
Building 8	x axis	49	54	50	45
	y axis	51	53	66	64
Building 9	x axis	50	54	58	54
	y axis	46	44	50	52
Building 10	x axis	45	50	53	48
	y axis	29	27	35	37
Building 11	x axis	30	35	38	33
	y axis	17	12	15	20
Building 12	x axis	41	51	51	41
	y axis	15	15	19	19
Building 13	x axis	52	59	59	52
	y axis	10	10	17	17
Building 14	x axis	60	64	70	66
	y axis	17	14	20	23
Building 15	x axis	70	74	80	76
	y axis	27	24	30	33
Building 16	x axis	53	62	62	53
	y axis	32	32	40	40
Building 17	x axis	62	68	68	62
	y axis	36	36	43	43

		Lower Left	Lower Right	Upper Left	Upper Right
Building 18	x axis	56.5	59.5	63	60
	y axis	44.5	43	49	50.5
Building 19	x axis	61	66	69	64
	y axis	56	54	60	62
Building 20	x axis	66	70	76	72
	y axis	52	50.5	62.5	64
Building 21	x axis	71	79	81	73
	y axis	70	66	70	74
Building 22	x axis	84	91	93	86
	y axis	63	59	63	67
Building 23	x axis	83.5	89.5	92	86
	y axis	74	70.5	75.5	79
Obstacle 1	x axis	56	100	100	58
	y axis	63	92	100	100
Obstacle 2	x axis	15	46	45	28
	y axis	77	90	100	100
Obstacle 3	x axis	0	21	21	0
	y axis	86	99	100	100
Obstacle 4	x axis	0	20	13	0
	y axis	21	32	74	63
Obstacle 5	x axis	4	22	40	33
	y axis	20	15	30	36
Obstacle 6	x axis	0	60	60	0
	y axis	0	0	2	18
Obstacle 7	x axis	54	100	100	100
	y axis	0	0	46	45.5
Obstacle 8	x axis	72	76	89	81
	y axis	41	40	57	61

APPENDIX F. COORDINATES FOR INDOOR OBSTACLES

		Lower Left	Lower Right	Upper Left	Upper Right
Wall 1	x axis	0	31	31	0
	y axis	18	18	22	22
Wall 2	x axis	28	32	32	28
	y axis	20	20	30	30
Wall 3	x axis	28	32	32	28
	y axis	40	40	60	60
Wall 4	x axis	0	30	30	0
	y axis	48	48	52	52
Wall 5	x axis	28	32	32	28
	y axis	70	70	80	80
Wall 6	x axis	29	60	60	29
	y axis	78	78	82	82
Wall 7	x axis	58	62	62	58
	y axis	30	30	100	100
Wall 8	x axis	59	70	70	59
	y axis	38	38	42	42
Wall 9	x axis	80	100	100	80
	y axis	38	38	42	42
Wall 10	x axis	79	100	100	79
	y axis	73	73	77	77
Wall 11	x axis	78	82	82	78
	y axis	75	75	80	80
Wall 12	x axis	78	82	82	78
	y axis	90	90	100	100

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- Abraham, Ajith, and Vitorino Ramos. 2003. "Web Usage Mining Using Artificial Ant Colony Clustering and Linear Genetic Programming." In *The 2003 Congress on Evolutionary Computation 2*: 1384–1391.
<https://doi.org/10.1109/CEC.2003.1299832>.
- Charlier, Bernadette. 1995. "The Greedy Algorithms Class: Formalization, Synthesis and Generalization." Lecture at UCL, Belgium.
<https://pdfs.semanticscholar.org/0375/63cfe4d9bed811b49b7eed202ac80e497b2.pdf>.
- Beni, Gerardo. 2005. "From Swarm Intelligence to Swarm Robotics." In *Swarm Robotics*, edited by Şahin Erol and Spears William M, 1–9. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-30552-1_1.
- Blanchard, Benjamin S., and Wolter J. Fabrycky. 2011. *Systems Engineering and Analysis*, 5th ed. Upper Saddle River, NJ: Prentice Hall.
- Blum, Christian, and Xiaodong Li. 2008. "Swarm Intelligence in Optimization." In *Swarm Intelligence: Introduction and Application*, edited by Christian Blum and Daniel Merkle, 43–85. Berlin, Heidelberg: Springer Berlin Heidelberg.
https://doi.org/10.1007/978-3-540-74089-6_2.
- Choset, Howie. 2000. "Coverage of Known Spaces: The Boustrophedon Cellular Decomposition." *Autonomous Robots* 9, no. 3 (December): 247–253.
<https://doi.org/10.1023/A:1008958800904>.
- Dorigo, Marco, Vittorio Maniezzo, and Alberto Coloni. 1996. "Ant System: Optimization by a Colony of Cooperating Agents." *IEEE Transactions on Systems, Man, and Cybernetics Part B (Cybernetics)* 26, no. 1 (February): 29–41.
<https://doi.org/10.1109/3477.484436>.
- Feng, Emily, and Charles Clover. 2017. "Drone Swarms vs. Conventional Arms: China's Military Debate." Last modified August 24, 2017.
<https://www.ft.com/content/302fc14a-66ef-11e7-8526-7b38dcaef614>.
- Gage, Douglas W. 1995. "UGV History 101: A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts." *Unmanned Systems Magazine* 13, no. 3 (January): 9–32. <http://www.dtic.mil/dtic/tr/fulltext/u2/a422845.pdf>.
- Galceran, Enric, and Marc Carreras. 2013. "A Survey on Coverage Path Planning for Robotics." *Robotics and Autonomous Systems* 61, no. 12 (August): 1258–1276.
<https://doi.org/10.1016/j.robot.2013.09.004>.

- Glenn, Russell W. 1996. *Combat in Hell: A Consideration of Constrained Urban Warfare*. Washington, DC: RAND.
- Goldman, Joshua. 2016. "Street View Your Life: These Are the 360-Degree Cameras Coming for 2016." CNET. Last modified February 23, 2016. <https://www.cnet.com/news/360-degree-cameras-2016/>.
- Goss, Simon, Serge Aron., Jean L. Deneubourg, and Jacques M. Pasteels. 1989. "Self-Organized Shortcuts in the Argentine Ant." *Naturwissenschaften* 76, no. 12 (December): 579–581. <https://doi.org/10.1007/BF00462870>.
- Kennedy, James, and Russell C. Eberhart. 1995. "Particle Swarm Optimization." In *Proceeding of the IEEE International Conference on Neural Networks* 6: 1942–1948. <https://doi.org/10.1109/ICNN.1995.488968>.
- Morin, Pascal, and Claude Samson. 2004. Trajectory Tracking for Non-Holonomic Vehicles. In *Robot Motion and Control*, edited by Krzysztof Kozłowski, 3–23. London: Springer. https://doi.org/10.1007/978-1-84628-405-2_1.
- Naffin, David J., and Gaurav S. Sukhatme. 2004. "Negotiated formations." In *Proceedings of the Eighth Conference on Intelligent Autonomous Systems*: 181–190. <https://pdfs.semanticscholar.org/c5f8/c452592054817793d25e1b573c3e377ee8de.pdf>.
- Omran, Mahamed, Andries P. Engelbrecht, and Ayed Salman. 2005. "Particle Swarm Optimization for Image Clustering." *International Journal of Pattern Recognition and Artificial Intelligence* 19, no. 3 (May): 297–321. <https://doi.org/10.1142/S0218001405004083>.
- Raza, Muhammad. 2018. "Minimize Function using Particle Swarm Optimization." Mathworks. Last modified June 21, 2017. https://www.mathworks.com/matlabcentral/fileexchange/67804-particle-swarm-optimization-pso-matlab-code-explanation?s_tid=prof_contriblnk.
- Robot Platform. n.d. "Robot Locomotion." Accessed April 29, 2018. http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html.
- Shi, Yuhui, and Russell C. Eberhart. 1998. "A Modified Particle Swarm Optimizer." In *Proceedings of the IEEE Conference on Evolutionary Computation* 6: 69–73. <https://doi.org/10.1109/ICEC.1998.699146>.
- Toksari, M. Duran. 2007. "Ant Colony Optimization Approach to Estimate Energy Demand of Turkey." *Energy Policy* 35, no. 8 (August): 3984–3990. <https://doi.org/10.1016/j.enpol.2007.01.028>.

- Ujgin, Supiya, and Peter J. Bentley. 2003. "Particle Swarm Optimization Recommender System." In *Proceedings of the IEEE Swarm Intelligence Symposium*: 124–131. <https://doi.org/10.1109/SIS.2003.1202257>.
- United Nations, Department of Economic and Social Affairs, Population Division. 2014. *World Urbanization Prospects: The 2014 Revision, Highlights (ST/ESA/SER.A/352)*. <https://esa.un.org/unpd/wup/publications/files/wup2014-highlights.pdf>.
- Ünler, Alper. 2008. "Improvement of Energy Demand Forecasts Using Swarm Intelligence: The Case of Turkey with Projections to 2025." *Energy Policy* 36, no. 6 (June): 1937–1944. <https://doi.org/10.1016/j.enpol.2008.02.018>.
- U.S. Department of Defense, Robotic Systems Joint Project Office. 2011. *Unmanned Ground Systems Roadmap*. Washington, DC. <http://www.dtic.mil/dtic/tr/fulltext/u2/a570570.pdf>.
- Wang, Han, Muqing Cao, Hao Jiang, and Lihua Xie. 2018. "Feasible Computationally Efficient Path Planning for UAV Collision Avoidance." Paper presented at IEEE 14th International Conference on Control and Automation, Anchorage, AK.
- Zelinsky, Alexander, Ray Jarvis, Jennifer Byrne, and Shin'ichi Yuta. 1993. "Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot." In *Proceedings of International Conference on Advanced Robotics* 13: 533–538. <http://pinkwink.kr/attachment/cfile3.uf@1354654A4E8945BD13FE77.pdf>.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California