אוניברסיטת בר-אילן

כיסוי שטח ע"י מערכת מרובת רובוטים

עידו איכר

אוניברסיטת בר-אילן, רמת-גן, ישראל 2007

תקציר

כיסוי שטח הוא תחום חשוב אותו ניתן למצוא בהרבה אפליקציות חיוניות. עבוא אפליקציות אלו קבוצה של רובוטים זהים ולא יקרים אשר יכולים לבצה כיסוי שטח יכולה להיות פתרון הולם. לדוגמא עבור שמירה על בניינים אשר בדרך כלל מעורבת בפטרולים ע"י בני אדם, שימוש במערכת כזו יכול להיות פתרון טוב.

הרבה מחקר נעשה בתחום כיסוי שטח ע"י מערכת מרובת רובוטים. הרבה מהאלגוריתמים הקיימים מסתמכים על תקשורת וסנסורים מתקדמים אשר משמשים ליצירת מיפוי השטח ולוקליזציה, אבל בעיות כגון איבודי תקשורת, בעיות לוקליזציה בעקבות אי דיוק של סנסורים ועוד בעיות היכולות לצוף בעולם האמיתי עדיין לא נחקרו וטופלו עד הסוף.

אנחנו מציגים אלגוריתם חדש עבור מערכות מרובות רובוטים אשר משמש בתור קופסה שחורה אשר מיועדת לעבוד עם אלגוריתמים לכיסוי שטח קיימים במטרה למקסם את כיסוי השטח ולמזער את סטיית התקן המתקבלת על פני הפעלות מרובות. האלגוריתם שלנו לא משתמש בלוקליזציה ולכן זקוק רק לסנסורים נפוצים אשר ניתן למצוא בהרבה מהפלטפורמות הרובוטיות הנמצאות היום בשוק, זאת אומרת שרק שינויי חומרה זניחים נחוצים להתבצע על מנת לתמוך באלגוריתם שלנו. האלגוריתם שלנו אינו מסתמך על פרוטוקולי תקשורת בין הרובוטים ולכן אינו רגיש לאיבודי תקשורת. אנחנו מבצעים ניסויים מרובים עבור האלגוריתם הכוללים סביבות ריצה שונות, מספר משתנה של רובוטים, שגיאות מסומלצות אשר עלולות בעולם האמיתי ועוד. אנחנו מראים שאפליקציות רובוטיות קיימות יכולות להשתמש באלגוריתם שלנו ולשפר את ביצועי כיסוי השטח.

Acknowledgments

I would like to thank Dr. Gal Kaminka who was my supervisor.

His patience and ongoing guidance is what ensured the completion of this thesis. Thank you for being my mentor, guide and friend.

Abstract

Area coverage is an important task which can be found in many essential applications. For these applications utilizing a large number of identical and inexpensive robotic platforms, which are able to perform area coverage, may provide an appropriate solution. For example, security tasks in pubic or government buildings, usually involve human patrol, using a multi-robot system would have been good.

Much work has been done in the area of multi robot coverage. Many of the existing algorithms rely on communication and advanced sensors used for maps creation and localization but problems such as communication loss, localization problems due to sensors inaccuracy and many other problems which could occur in the real world are not fully addressed.

We present a new robust online algorithm for multi-robot coverage which behaves as a black box designed to work with single-robot coverage existing algorithms. Our algorithm maximizes the area coverage and minimizes the standard deviation over multiple operations for these algorithms. Our algorithm does not perform any localization and therefore requires only few common sensors which can be found in many of today's robotic platforms, this means no hardware modifications to the robots are required in order to support our algorithm. Our algorithm does not rely on communication protocols between the robots and is therefore not sensitive to any communication losses. We perform extensive experiments for our algorithm which includes different environments, different number of robots, simulated errors which might occur in the real world etc. We show that real robotic platforms can benefit from using our algorithm.

Contents

1 Introduction

Area coverage is an important task which can be found in many essential applications such as land mine deployment [1], ship hull cleaning [2], planetary surface exploration [3], [4], [5], floor cleaning [6], de-mining [7] and more.

In these applications the robot is placed in a bounded work area, most likely to contain obstacles, and equipped with some kind of a tool (for example a robotic vacuum cleaner will probably be equipped with a suction nozzle) that must visit every point inside the work area.

Area coverage appears to be a task perfectly matched for a multi robot approach in the following aspects:

- The target area might be very complex for robots.

  Robots may encounter many types of obstacles from all kinds, and the possibility of a robot getting stuck does exist. The use of multiple robots allows the mission to be performed by the remaining robots.
- Many separated areas must be dealt with. The use of many robots allows these areas to be covered in parallel, rather than one at a time.

We would like our multi-robot algorithms to be **complete** (complete coverage guarantees that the robots pass over all reachable points in the target environment), **efficient** (efficiency is evaluated in terms of area coverage over time) and robust, in the way that most of the robot failures along the way can be overcome.

We present a comparative evaluation of well known single-robot algorithms (Parallel, Random and Spiral) and show that they suffer from significant weaknesses when applied. We then present a new online algorithm which is actually a kind of a black box designed to work with any other existing algorithms. It does not use any localization or maps, only limited communication between robots. We add our algorithm to the previously tested algorithms, repeat the comparative evaluation and show that our algorithm improves the area coverage results by:

- Maximizing the area coverage:

  We show that when our algorithm is being used, robots cover more area in less time.
- Minimizing the standard deviation over multiple operations:

  We show that using our algorithm leads to a significant reduction in the variance of area coverage, in repeated applications.

We perform extensive experiments which include different environments with different number of obstacles. We vary the number of robots from 1 to 10. We simulated different errors which might occur in the real world such as odometer errors and test their influence on our algorithm's performance.

2 Background and Related work

A lot of work has been performed in the area of single and multi robot coverage. A recent survey by Choset regarding area coverage algorithms can be found in [8]. There are two basic types of algorithm: online algorithms which are algorithms that do not need any prior knowledge of the work area, and offline algorithms that assume the robot has a prior knowledge of how the work area looks like which for some applications is unrealistic. Complete algorithms guarantees that the robot passes over all reachable points in the target environment. We will consider an algorithm to be robust if this algorithm is not affected by failures. For example in a robust algorithm, when one of the robots breaks down we expect that the other robots will cover for him and take over his work.

Balch and Arkin used heuristic approaches for investigating multi robot coverage tasks [10, 11], and one of the heuristics they used, repulsion from other robots ensured that robots spread out over the environment resulting a better area coverage. In our algorithm, the repulsion is not from other robots but from cells which are already handled by other robots.

In [12] Warner et al. presents a cellular decomposition algorithm for a group of robots which uses chemical traces which evaporate with time to communicate between the robots. In our algorithm we also divide the work area into cells and each robot communicates by picking up the IR transmissions of other robots, unlike the chemical traces the IR signal does not stay, it gives a real time indication of robots in the same cell.

In [13] M. Jager and B. Nebel present a robust algorithm for multiple robots cleaning. They divide the room into polygons, and then the robots allocate and clean these polygons. To allocate a polygon means that a robot intends to clean the polygon and announce this. It is not assumed that the robots are always able to communicate with each other. Our algorithm does not require any communication between the robots and therefore more robust since it can not be influenced by communication problems.

In [14] Spires and Goldsmith shows an off-line multi-robots algorithm based on cellular decomposition. This algorithm guarantees a robust coverage but unfortunately, it works only in obstacle-free work-areas. Our algorithm can handle obstacles. Spires and Goldsmith show that the initial positions of the robots within the work-area affects the coverage time. Our algorithm's area coverage results are not affected by the initial location of the robots.

In [15] Kurbayashi et al. introduce an off-line multi-robot coverage algorithm. This algorithm is based on a cellular decomposition. However, no guarantees on robustness are provided. Our algorithm is also based on cellular decomposition but provides complete robustness.

Hazon and Kaminka [17] and later Agmon, Hazon, and Kaminka [16], and Hazon, Mielli and Kaminka [18] present a family of algorithms for multi-robot coverage, based on spanning trees that cover an approximately cell-decomposed area. These algorithms guarantee complete, efficient coverage, and are robust to catastrophic failures of robots: As long as one robot is alive, the coverage will complete. However, their algorithms all rely on precise positioning within a grid decomposition of the work area, and reliable, unlimited communications about locations and covered areas. In contrast, our work focuses on a heuristic approach which is not guaranteed to succeed in covering the target area, but makes no assumptions as to localization, and utilizes simpler communications (only robot IDs).

3 Comparative evaluation of coverage algorithms

In this chapter we present a comparative evaluation of three well known single-robot algorithms: parallel, random and spiral. This comparative evaluation is actually a preparation for the next stage where we add our algorithm to each one of these three algorithms and show that an improvement has been achieved in the coverage performance for each one of the algorithms. Comparing between these three algorithms is interesting also because it has never been done before.

The experiments (Section 5.2) show that there is no statistical significant indication that the random performance is better than the spiral or vice versa, but there is a statistical significant indication that both random and spiral are better than the parallel.

3.1 Parallel algorithm

The first algorithm we chose is the parallel algorithm (Algorithm 1) which is used for applications such as lawn mowing. It is used both in exact and approximate cell decomposition and in the FriendlyVac vacuum-cleaning robot by Friendly Robotics, Ltd. The algorithm accepts as input a parameter determining the distance between adjacent parallel legs. Typically, this is set to the width of the robot's tool.

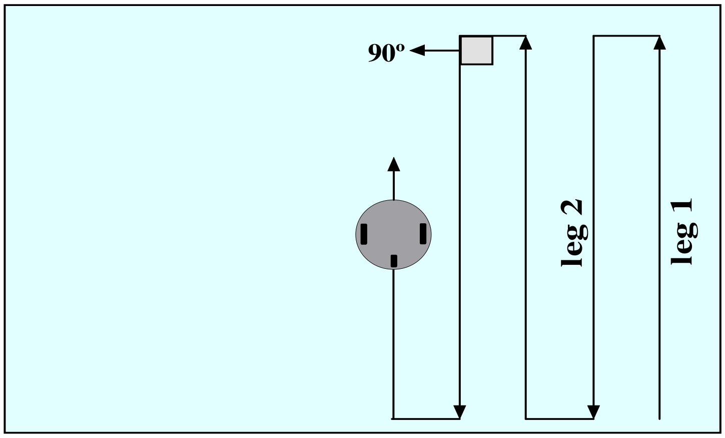


Figure 3.1: Parallel coverage

Algorithm 1 Parallel coverage

```
1:  Let Advancement_Distance ⟵ The robot's advancement distance between
    every two parallel legs (in our case this distance will be in the size of the
    tool our robot is carrying)
2:  while barrier is not reached do
        Perform a forward leg called leg1
3:  Perform a 90º turn to the left
4:  while Advancement_Distance is not reached do
        Move forward
5:  Perform a 90º turn to the left
6:  while barrier is not reached do
        Perform a forward leg called leg2 which is parallel to leg1
7:  Perform a 90º turn to the right
8:  while Advancement_Distance is not reached do
        Move forward
9:  Perform a 90º turn to the right
10: Goto 2
```

3.2 Random algorithm

The second algorithm we chose is the random algorithm (Algorithm 2). This algorithm works as follows. An arbitrary direction is picked, and then forward motion is initiated until the robot is blocked. Then, a new direction is selected, and the process is repeated.

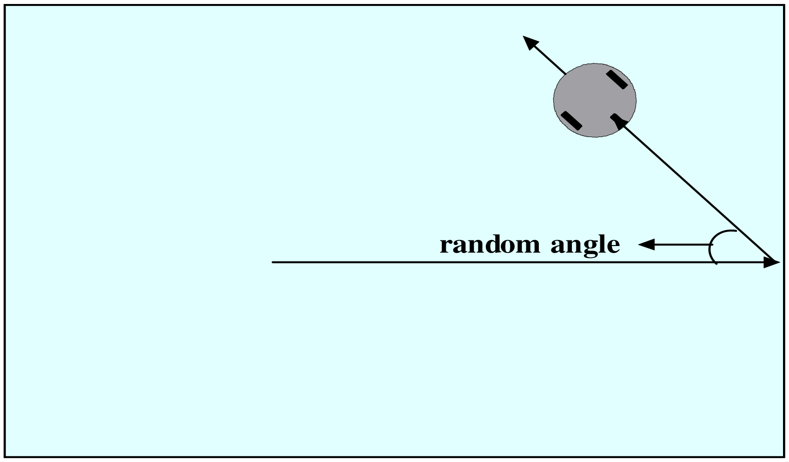


Figure 3.2: Random coverage

Algorithm 2 Random coverage

```
1:  while barrier is not reached do
        Perform a forward leg
2:  Perform a turn to a random direction
3:  Goto 1
```

3.3 Spiral algorithm

The third algorithm we chose is the spiral algorithm (Algorithm 3). This scan is actually a combination of random scan and spiral behavior. The spiral behavior will only be triggered in open areas after a long leg is detected since it is not effective in small areas. A variation of this algorithm is used in iRobot's Roomba vacuum-cleaning robot.

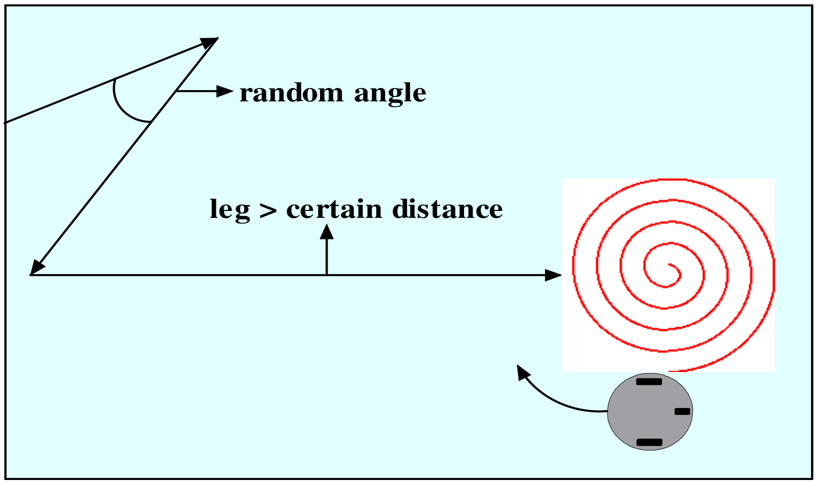


Figure 3.3: Spiral coverage

Algorithm 3 Spiral coverage

```
1: while leg distance does not exceed a certain distance do
       Perform random algorithm
2: while barrier is not reached do
       Perform spiral movement with overlapping circle movements
3: Goto 1
```

3.4 Results

The results show that the single-coverage algorithms, when used in multi-robot settings, have a large variance. The next chapter addresses this, and shows how to improve these results.

4 Our Algorithm

There were many criterions we had to consider while developing our algorithm. Cost efficiency is a key element especially when dealing with multi robot systems. In some application, large number of simple cheap robots may be the answer while in others the right solution may be much smaller number of robots with higher sensors, effectors, processing and communication capabilities. As technology continues to evolve we can see that the growing diversity of sensors and effectors technologies results a growing demand for integrating "off the shelf" sensors into robots. When possible, it will pay to "piggy back" on systems previously developed for other applications, or on mass market items which have been developed and cost-optimized for other purposes.

We decided to embrace the "piggy back" method and develop an algorithm that could be integrated into many of the robotic platforms that can be found in today's market, platforms which have already been cost-optimized. We created an algorithm which can operate by solely using common sensors which is actually a kind of a black box designed to work with any other existing online coverage algorithm. Here, online means coverage is performed without using any map of the environment which makes it more robust and suitable for real world applications.

We believe that in order to achieve good coverage as a team, robots must spread out over the environment, otherwise their coverage areas overlap, hurting overall performance. Our algorithm assumes that the work area is divided into cells, and uses heuristics to decide in real time how many robots are required in a cell, how long should a robot work in a cell before moving to the next one, etc, to spread the robots over the entire work area and improves the coverage performance.

4.1 Sensors

We would like our algorithm to be able to function solely by using the most common sensors which can be found in many of today's robotic platforms. The reason we are motivated to do so is to create an algorithm that will allow converting these robots from a single robot application to a multi robot application solely by changing the robot's software and without performing any hardware changes (or performing only minor insignificant hardware changes).

The decision to use only common sensors makes this research a lot more challenging since we only use the resources provided by the existing robotic platforms designed for single robot applications to support multi robot applications.

4.1.1 Sensors considerations

When designing a multi robot system, choosing the sensors is one of the most important tasks. There is no doubt that in an ideal world we would want our robots to know every possible thing about each other and the world, since better decisions can be made if every robot has an accurate map of the area it is supposed to cover, knowledge of its current location and where it has been, and unlimited communication with the other robots so that every robot will know all the above about all the other robots.

Theoretically, if we could have this kind of multi robot system we could reach optimal area coverage results. However, when we examine many of today's robotic platforms we see that none of them relies on the need to know their exact location and none uses an offline algorithm which requires a map of the work area. The reason for this is because achieving these capabilities is just too expensive. We should keep in mind that systems developed for mass-production are a lot more tighter in budget than specially designed systems which are usually designed to forefeel the needs of a specific customer such as military systems for example where money is not always such an issue, and since we want our algorithm to be relevant for mass-marketed robots we have to rely on the most common sensors found in these kind of robotic platforms.

4.1.2 Sensors requirements

Our algorithm requires a few essential sensors capabilities:

Virtual borders

We need the ability to create virtual borders in order to detect transitions between different areas (cells, as we call them). For example if we are planning to cover an entire hotel floor, we should have the ability to detect whenever we enter or leave a room. Many of today's robotic platforms come with virtual border capability in order to allow the user to prevent the robot from accessing certain areas.

Robots ID

Our algorithm assigns each robot with an ID number which rises from 1 to n (where n is the total number or robots). Robots should have the ability to transmit their ID and receive others. We want robots that work in the same cell to detect each others ID as fast as possible because the faster they will be aware of one another the better results our algorithm will produce. We examined two optional ways for assembling the ID transmitter (Figure 4.1). In the first option the transmitter is aimed towards the side of the robot and in the second it is aimed towards the front of the robot.

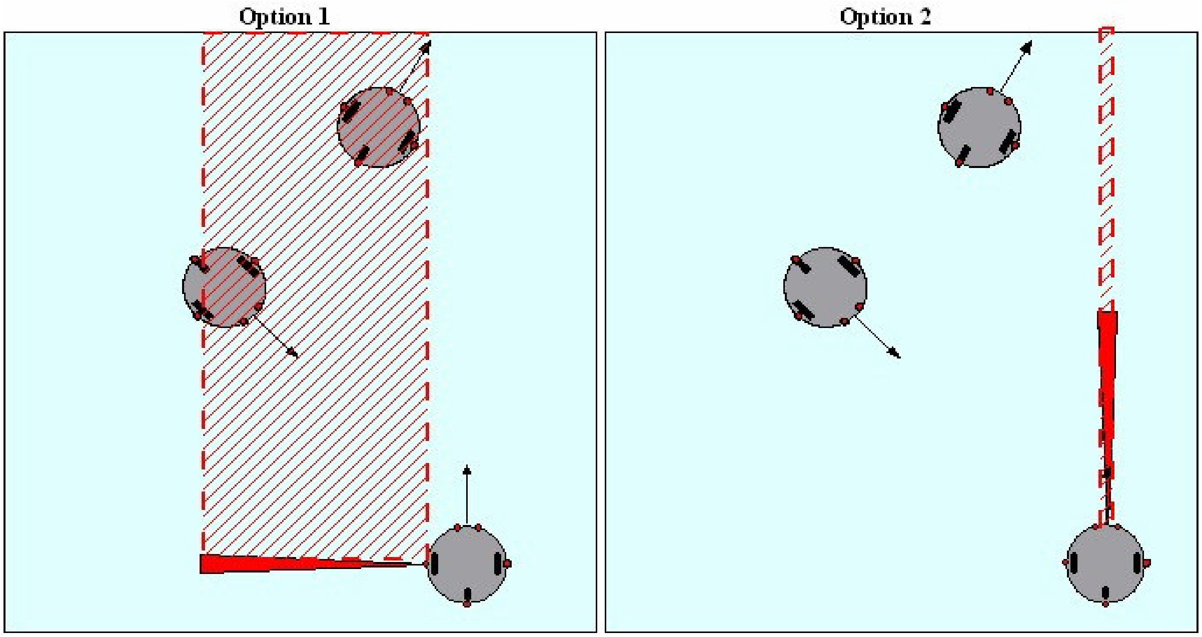


Figure 4.1: Optional ID transmitter assembling

We decide to go with the first option because as we can see in Figure 4.1, the area covered by the transmitter in option 1 is bigger than the one in option 2, therefore using a side transmitter will allow robot to detect each other more quickly.

4.2 Behaviors

Our algorithm is based on four behaviors: **Count Off**, **Scan**, **Introduction** and **Search.** The high level of our algorithm which is responsible for deciding what behavior should be performed and when will look as followed:

Algorithm 4 High level

```
1:  Perform the Count-Off algorithm // Algorithm no. 5
2:  Perform the Scan algorithm // Algorithm no. 6
3:  Perform the Search algorithm // Algorithm no. 10
4:  Goto 2
```

4.2.1 Count-Off behavior

The Count-Off behavior's goal is to find the total number of robots in the group. This behavior is performed only once at the beginning of the operation when we can still rely on the fact that all robots are grouped together.

The following figure (Figure 4.2) illustrates the Count-Off behavior for a group of 6 robots:

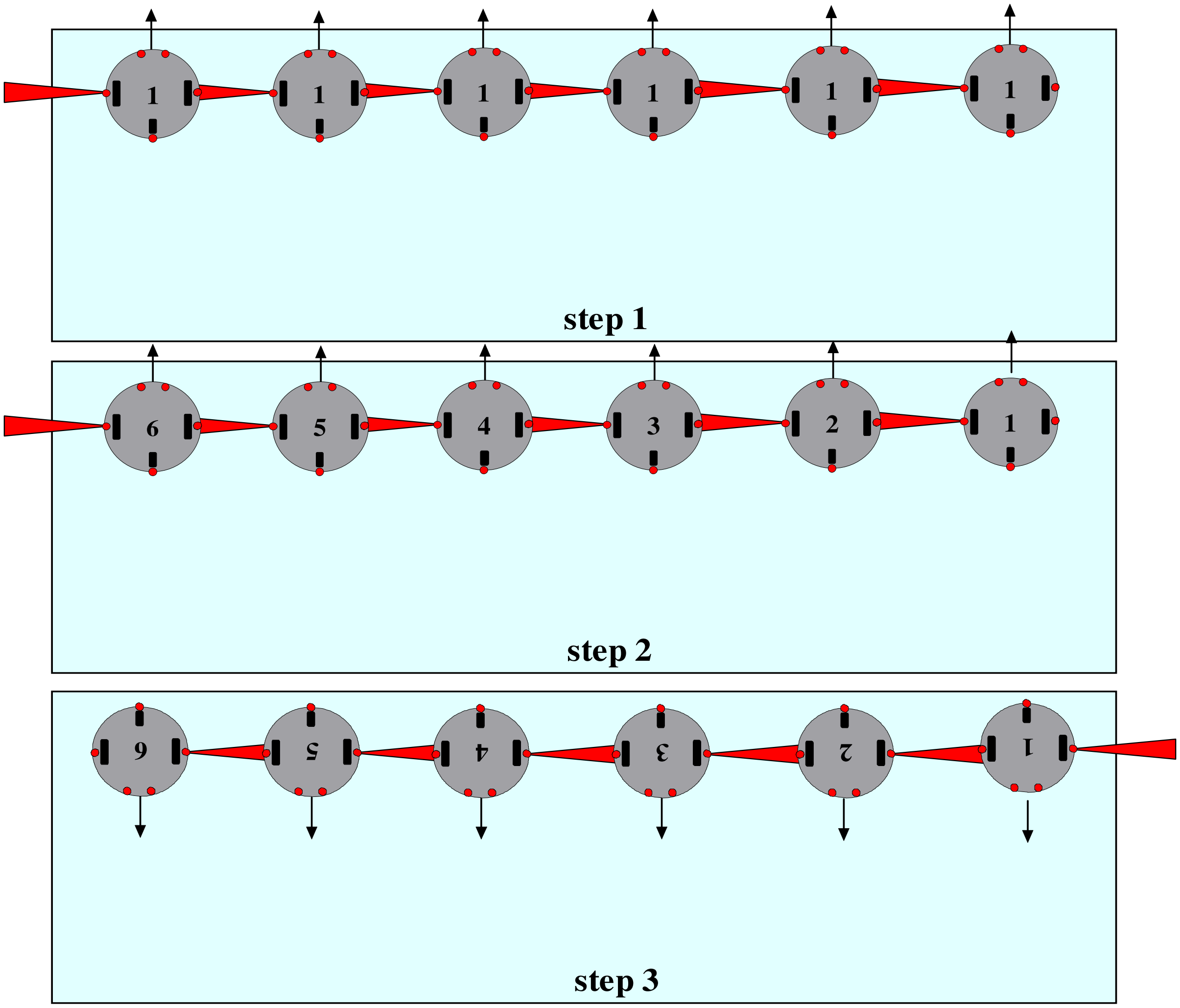


Figure 4.2: Count-Off behavior

- Step 1: In this step we see a group of robots in their initial state which is aligned in a straight line facing the wall with an ID of 1. Each robot detects the ID transmission of the robot on the side and follows the next rule: The personal ID of a robot will be equal to the received ID from the robot on the right (which is the only robot detected at this step) incremented by 1.
- Step 2: In this step we see that after following the rule described in step 1 we actually ensure that after a period of a few seconds (which is more than enough for each robot to receive its neighbor's transmission), all robots will

have a unique ID which starts from 1 for the robot on the right side and increases up to 6 for the robot on the left side.

- Step 3: In this step each robots performs a 180° turn which brings us to a situation where each robot detects the ID transmission of the robot on the other side from the one detected in step 1. After the turn is completed each robot will follow the next rule: A robot will transmit the maximal ID between its own personal ID and the received ID from his neighbor's transmission, and will save this maximal ID as the total number of robots in the group. Following this rule will ensure that after a period of a few seconds all the robots in the group will have the required information of the total number of robots.

Algorithm 5 Count-Off behavior

```
Let Personal_ID ⟵ The robot's personal ID
Let Received_ID ⟵ The received ID from some other robot transmission
Let Total_Num_Of_Robots ⟵ The total number of robots in the group
Let Reception_Time ⟵ Time it takes for each robot to receive its neighbor's transmission
Personal_ID ⟵ 1
while 5 seconds did not elapse do
    Transmit the Personal_ID value
    if Personal_ID < (Received_ID+1) then
        Personal_ID ⟵ (Received_ID+1)
Perform a 180° turn
while Reception_Time seconds did not elapse do
    Transmit the Total_Num_Of_Robots value
    Total_Num_Of_Robots ⟵ Maximum(Personal_ID , Received_ID)
Stop
```

4.2.2 Scan behavior

The Scan behavior is responsible for the local area coverage inside the cells. During this behavior the robot will constantly transmit its ID. Any type of coverage algorithm can be used at this stage. This is shown in the Experiments chapter (Chapter 5) where we test our algorithm using other algorithms such as parallel, random and spiral. The fact that we allow any type of algorithm to be used for the local cell area coverage makes our algorithm relevant for a wider range of robotic applications which currently use a specific algorithm and act as a single robot application and can be easily transformed into multi robot application by using our algorithm.

Algorithm 6 Scan behavior

```
Let Personal_ID ⟵ The robot's personal ID
Let Received_ID ⟵ The received ID from some other robot transmission
Let Required_Work_Time ⟵ The required time robot should work in the cell
Let Cell_Coverage_Algorithm ⟵ Any type of coverage algorithm which can
    be used for cell area coverage (parallel, random, spiral etc…)
Let Cells_Exist ⟵ An indication if there are cells in the current work area
Personal_ID ⟵ 1
while Cell_Coverage_Algorithm is performed do
    if Cells_Exist =True then
        Transmit the Personal_ID value
    if an ID transmission is received then
```

```
11:        Perform the Robot ID algorithm // Algorithm no. 7
12:        if Personal_ID = Received_ID or a new Received_ID is detected then
13:            Perform Introduction behavior
14:        if Required_Work_Time is reached and Cells_Exist =True then
15:            Goto 16 // End of Scan beahvior
16:  Stop
```

Our algorithm requires that each robot will collect certain information about the other robots and the environment. This information is essential for our algorithm functionality and will contain the following data:

Robots ID

When the Scan behavior starts each robot will be assigned with an initial ID equal to 1. When a robot detects some other robot's transmission with the same ID it will immediately increment its own ID by 1, this technique will ensure that each robot will have a unique ID over time.

The following figure (Figure 4.3) illustrates the Robot's ID mechanism:

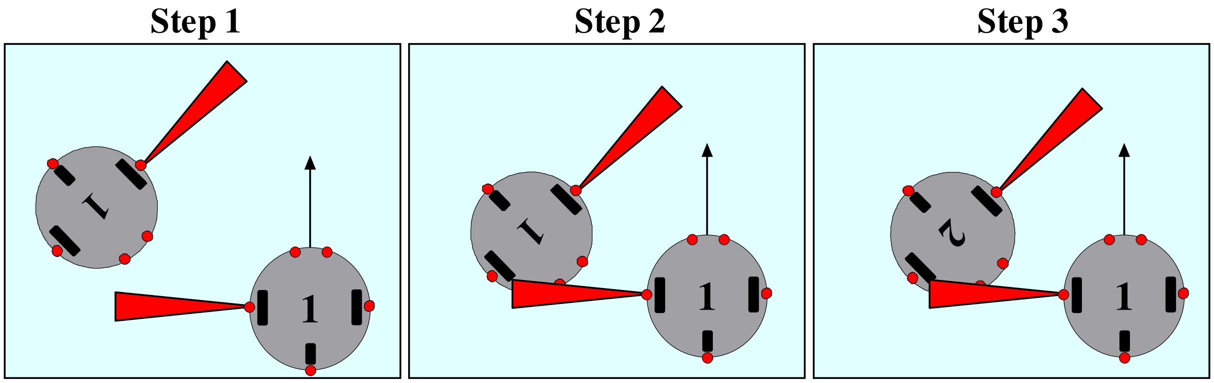


Figure 4.3: Robot ID mechanism

- Step 1: In this step we see two robots with the same ID 1 driving towards each other.
- Step 2: In this step we see that the left robot detects the ID transmission of the right robot (which is the same ID as its own)
- Step 3: In this step we see that the left robot reacts to this detection by increasing its ID to 2.

Algorithm 7 Robot ID

```
1: Let *Personal_ID* ⟵ The robot's personal ID
2: Let *Received_ID* ⟵ The received ID from some other robot transmission
3: **if** an ID transmission is received **then**
4:     **if** *Personal_ID* = *Received_ID* **then**
5:         *Personal_ID* ⟵ (*Personal_ID*+1)
6: Stop
```

Number of robots in the current cell

The Introduction behavior (Section 4.2.3) ensures that each robot will be assigned with a unique ID over time. This fact allows each robot to find the number of robots that are working in the same cell by following a simple rule: whenever a robot detects an ID transmission it will save the maximum between its personal ID and the received ID as the number of robots in the current cell.

Algorithm 8 Number of robots in the current cell

```
Let Personal_ID ⟵ The robot's personal ID
Let Received_ID ⟵ The received ID from some other robot transmission
Let Num_Of_Robots_In_Cur_Cell ⟵ The number of robots in the current cell
if an ID transmission is received then
    Num_Of_Robots_In_Cur_Cell ⟵ Maximum(Personal_ID,Received_ID)
Stop
```

Required work time

By analyzing the cell’s size and the number of robots that are working in the cell our algorithm estimate how much work time a robot should spend in the current cell. It's impossible to find one formula that will successfully estimate the required work time we have to spend in a cell in order to cover it for all the optional area coverage algorithms since every algorithm has its own special behaviors, and the estimation formula can not be successfully developed unless we know what these behaviors are up front. Therefore our black box will not try to achieve this information by itself but will be given this information as an input. This will require from whoever is integrating our black box with another algorithm to implement an estimation formula for the required work time. In order to understand this issue a bit better we will now show how we estimate the required work time for the parallel, random and spiral coverage algorithms.

During the operation we save the following information:

- *Average_Leg_Dist* - Average leg distance in cm
- *Average_Velocity* – Average robot velocity in cm/sec

- *Tool_Size* – The size of the tool our robot is carrying in cm

This following cell area estimation will give the best results when the shape of the cell is square which was good enough for us since many of the indoor areas which use robotic applications are types of rooms which in most cases are square. We calculate the cell area by using the following estimation formula:

Cell_Area = *(Average_Leg_Dist)* ^{2}

We calculate the required work time by using the following estimation formula:

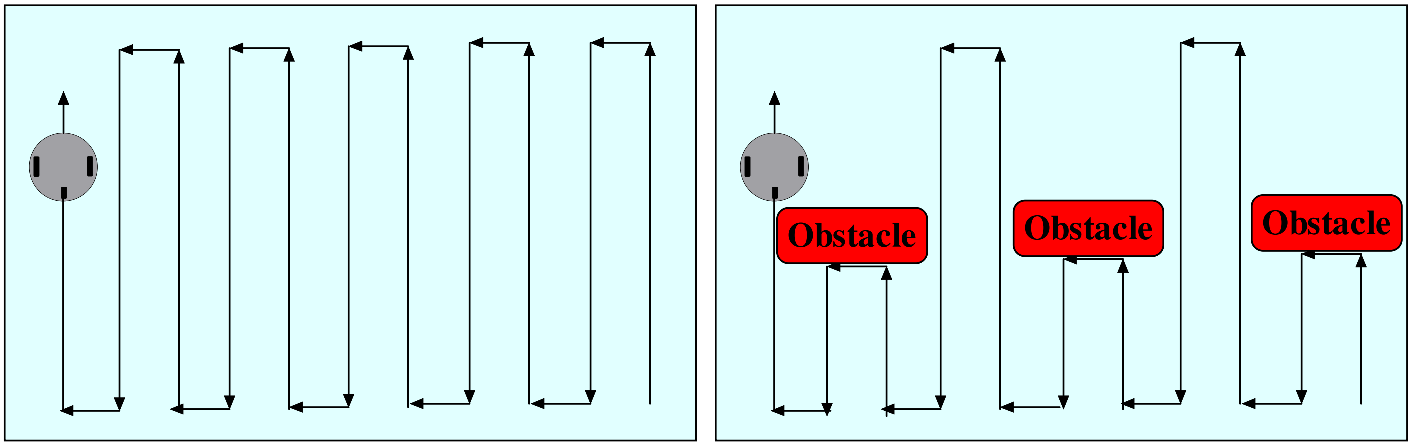
$$\boxed{\mathrm{Re}quiredWorkTime = \frac{\left(\frac{CellArea}{AverageVelocity \times ToolSize}\right)}{NumOfRobotsInCurCell}}$$

We did not settle for theoretical assumptions, and performed a sequence of tests to confirm our estimation formulas accuracy (Section 5.3).

In the real world the robot can encounter different obstacles and different cell shapes that can increase the cell complexity, and therefore influence the required work time. Here are some of the things that can influence the cell complexity:

- The number of obstacles we encounter

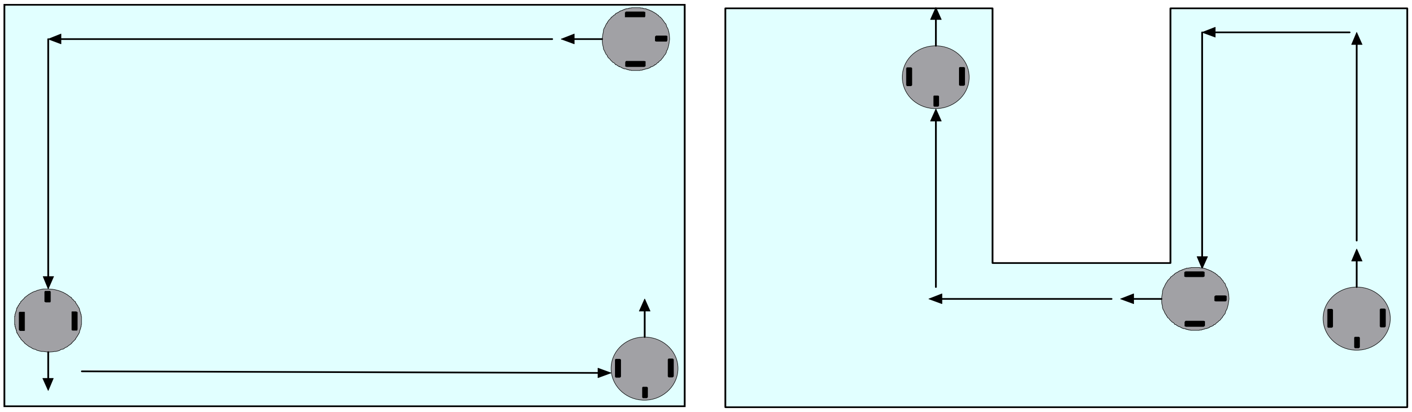
  If we take two identical cells and fill one of them with obstacles, it will take more time to cover the cell with the obstacles as we can see in the following figure (Figure 4.4):



Cell Without Obstacles **Cell With Obstacles**

Figure 4.4

- The number of right turns during the Search behavior in case wall following algorithm is used can also provide an indication about the complexity of the cell. Normal cell with a square shape will give us no right turn at all, the more right turns we detect the more complex the cell is as we can see in the following figure (Figure 4.5):



Cell without right turns **Cell with right turns**

Figure 4.5

Using this information can help us receive a better estimation formula for the required work time. But we should remember that the estimation formulas we presented are just a suggestion and there could probably be many other estimation formulas that will do the job nicely, so every algorithm that will use our black box will have to come up with its own estimation formulas.

4.2.3 Introduction behavior

The Introduction behavior’s goal is to make sure that each robot is assigned with a unique ID over time (when n robots are deployed in a cell their IDs will range from 1 to n over time) and to decrease the time it takes for robots that work in the same cell to be aware of one another existence, because the sooner robots will detect each other, the sooner the unnecessary robots will leave the cell and therefore save valuable work time. During this behavior the robot will constantly transmit its ID.

We should not assume that the ID transmitter transmits in a 360º sector, on the contrary, if we want to rely solely on common sensors we have to assume our transmitter only works in line of sight. Therefore, when a certain robot detects another one, it does not necessarily mean the opposite. To decrease the probability of a scenario where two robots are in the same area but only one of them detects the other from happening we created the Introduction behavior. When a robot performs the Introduction behavior it actually ensures that all robots within its radius and the range of its transmitter will detect its ID. Therefore, theoretically there will not be a scenario where one robot is aware of another robot's existence without the opposite. While robot is performing the Scan behavior it monitors all the other robots ID transmissions in the cell. When the received ID is identical to the robot's personal ID or when the received ID is a new one which hasn't been detected before, an Introduction behavior is triggered.

The following figure (Figure 4.6) illustrates the Introduction behavior:

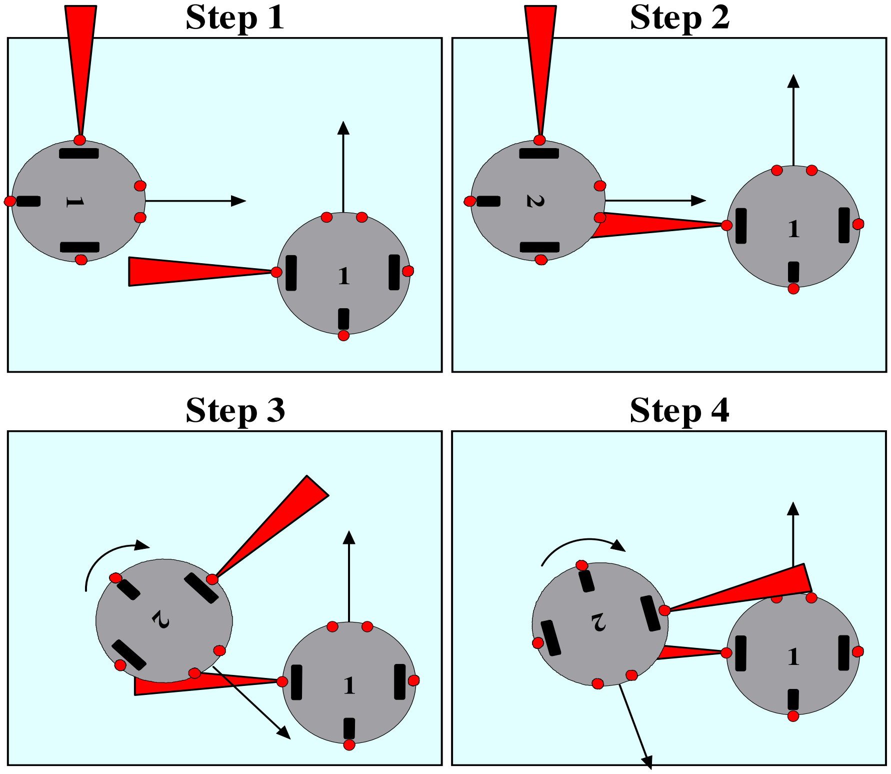


Figure 4.6: Introduction behavior

- Step 1: In this step we see two robots both with the same ID 1 driving towards each other.
- Step 2: In this step we see that the left robot detects the ID transmission of the right robot and since the ID of the detected robot is the same as its own, it immediately increments it's ID by 1 (from 1 to 2) and starts the Introduction behavior.
- Step 3: In this step we see that the left robot starts performing a 360◦ turn.

- Step 4: In this step we see that the left robot transmission has reached one of the right robot's ID receivers and now both robots had finished Introducing themselves.

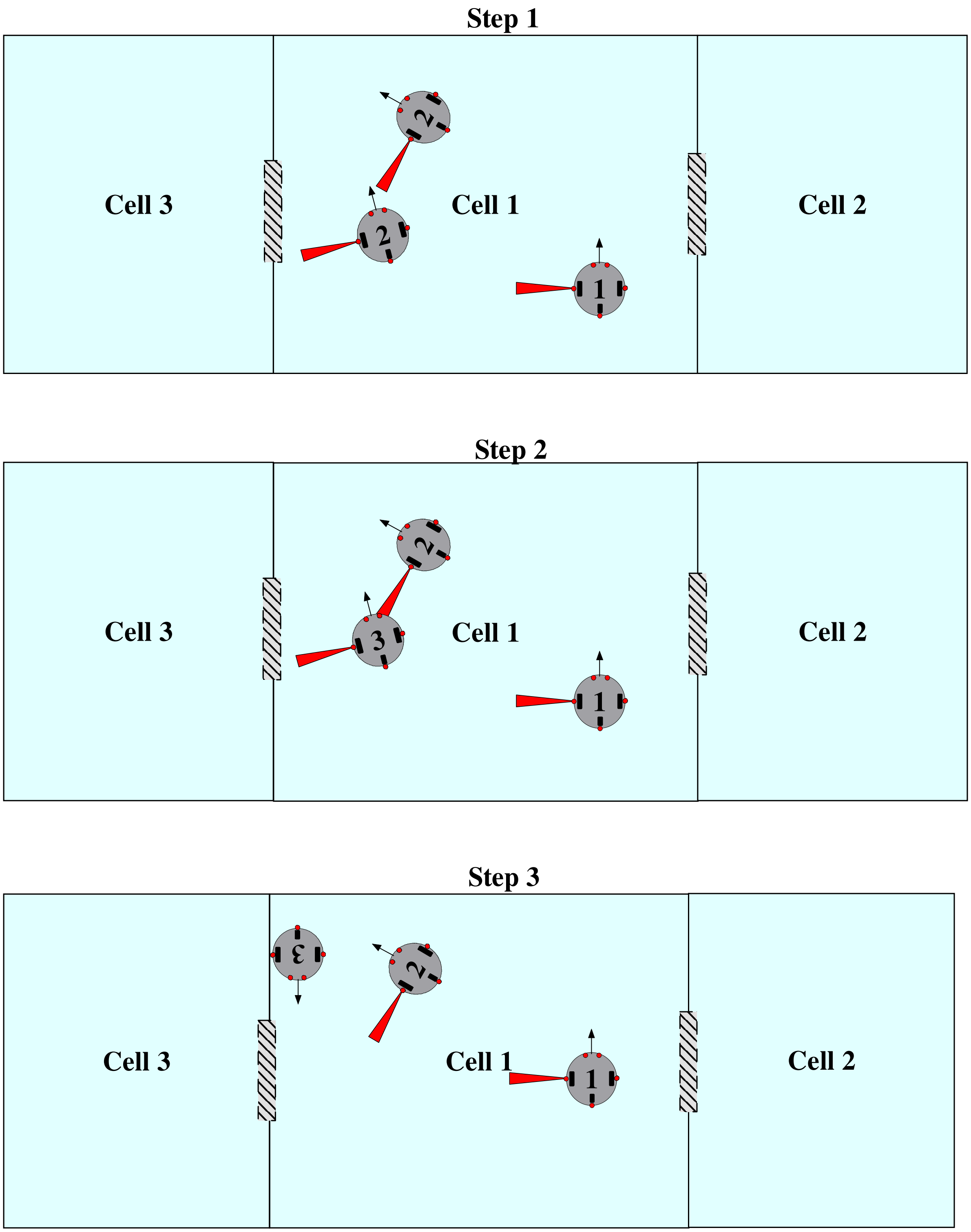
Algorithm 9 Introduction behavior

```
1: Let $Personal\_ID \longleftarrow$ The robot's personal ID
2: Let $Received\_ID \longleftarrow$ The received ID from some other robot transmission
3: if $Personal\_ID = Received\_ID$ then
4:     $Personal\_ID \longleftarrow (Personal\_ID+1)$
5: Perform a 360° turn
6: Stop
```

4.2.4 Search behavior

The Search behavior's goal is to bring us from the current cell to a new one. Once a robot decides it's time to leave the current cell or in other words when the Scan behavior ends, the Search behavior is triggered. During this behavior the robot will disable its ID transmission since it is no longer an active participator in the area coverage of the cell. Any type of search algorithm can be used at this stage. The fact that we allow any type of search algorithm to be used makes our algorithm relevant for a wider range of robotic applications which currently use a specific algorithm and act as a single robot application and can be easily transformed into multi robot application by using our algorithm.

The following figure (Figure 4.7) illustrates a Search Behavior which uses a wall following algorithm as the search algorithm:



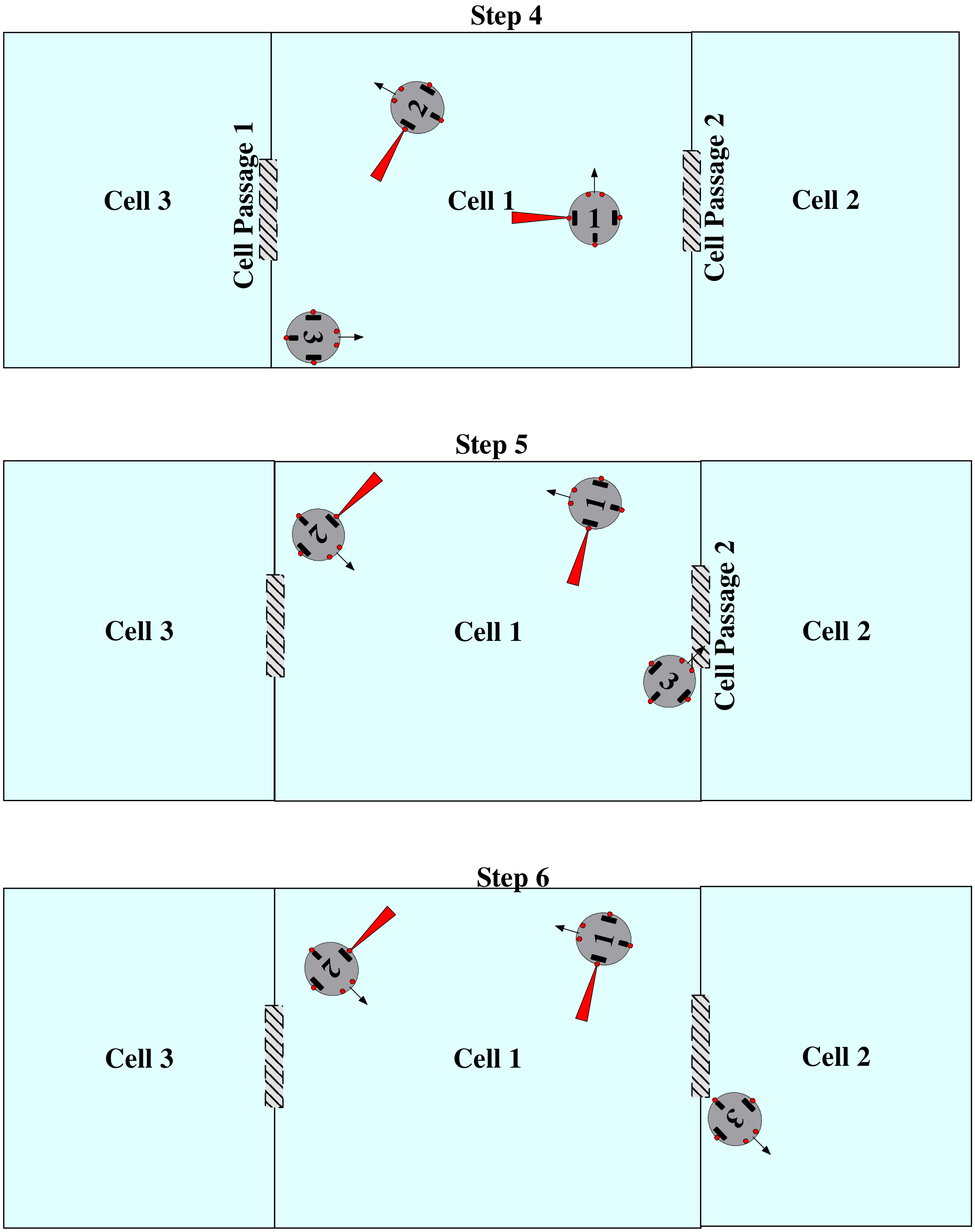

Step 4
Cell 3
Cell Passage 1
Cell 1
Cell Passage 2
Cell 2
1
2
3
Step 5
Cell 3
Cell 1
Cell Passage 2
Cell 2
1
2
3
Step 6
Cell 3
Cell 1
Cell 2
1
2
3

Figure 4.7

- Step 1: In this step we see a group of three robots performing the Scan Behavior inside Cell 1. There is one robot with ID 1 and another two robots with the same ID 2.
- Step 2: In this step we see that one of the robots with the ID 2 detects the other one with the same ID and increments its ID to 3.
- Step 3: In this step we see that the robot with ID 3 detects there are too many robots in the current cell and decides to trigger the Search Behavior in order to leave the current cell. It turns off its ID transmission and starts performing the wall following behavior in order to search for a cell passage.
- Step 4: In this step we see that the robot with ID 3 detects cell passage no. 1 which leads to cell 3 but decides to skip it. This is an example of how a robot should behave during the Search Behavior when a cell passage which was already used is detected. And in this example this robot had already passed through cell passage no. 1 which means that it had already covered cell 3, therefore it will keep on searching for a new cell passage which hasn't been used.
- Step 5: In this step we see that the robot with ID 3 detects cell passage no 2 which in this example is a new cell passage which hasn't been used by this robot, and therefore it decides to pass through it.
- Step 6: In this step we see the robot in the new cell.

Algorithm 10 Search behavior

```
Let Search_Algorithm ⟵ Any type of search algorithm which can be used
    for new cell search (wall following, etc…)
while Search_Algorithm is performed do
```

```
Disable ID transmission
if Cell passage is detected then
    Perform the new cell detection algorithm // Algorithm no. 11
    if New cell is detected then
        Stop
```

New cell detection

Our algorithm uses the robot's odometers to create a 2 dimensional grid, therefore every location has an [X,Y] representation. The cell passages size is a known value and will be determined by the maximal size of the cell passages that can be found in the work area and the odometers accuracy. For example if the work environment is an indoor terrain such as an office building, the cells are the offices and the cell passages are the doors to the offices then the cell passages size will probably be somewhere around 80 cm which is the size of a standard office door. Let's call the cell passages size *Cell_Passage_Size*. Every time a new cell passages is detected we will save its location. A cell passage is declared as new if the following exists: During the Search behavior when a cell passage is detected the robot will save its current location as [Xrobot,Yrobot] and compare it with all the other saved cell passages locations, if the distance between the robot's location and the location of all the other cell passages exceeds *Cell_Passage_Size* then it means that the current detected passage is a new one. For example lets assume that a certain robot has just detected a cell passage and its current location is [*Xrobot,Yrobot*], and until that point it has detected *N* number of new cell passages with the locations [*X1,Y1*], [*X2,Y2*], …, [*Xn,Yn*]. Let's look at distance *Dist* which is the formula for the distance between two points:

$$\boxed{Dist\,(i) = \sqrt{\left(Xrobot\ - Xi\right)^2 + \left(Yrobot\ - Yi\right)^2}}$$

For every *i=1…n* if *Dist(i)* is bigger then *Cell_Passage_Size*, then the current location [*Xrobot,Yrobot*] is a new cell passage location.

To prevent the scenario where a robot is performing an endless search for a new cell passage that does not exist because all the cell passages in the cell has been used before, we will follow the next rule: If a certain cell passage is detected for the second time during the Search behavior it means we are in an endless loop and there are no new cell passages in the current cell, in this case we should delete all the cell passages information of the current cell.

To prevent the scenario where a robot is performing an endless search for a new cell passage that does not exist because the current work area does not have any cells, we will follow the next rule: If we do not detect any cell passages during the Search behavior it means we are working in an area with no cells and therefore we should disable the Search behavior until a cell passage is detected during the Scan behavior, we should also stop transmitting the ID. Stopping the ID transmission is done in order to prevent other robots from detecting it, starting the Search Behavior and wasting valuable work time, and disabling the Search behavior is also done in order to save work time. The motivation for this improvement in the Search behavior is to make sure that when there are no cells in the work area, we will not damage the coverage performance by looking for cells that do not exist.

Algorithm 11 New cell detection

```
1:  Let New_Cell_Detect ⟵ The return value for this algorithm which indicates
```

```
    if the current detected cell passage is a new one. Return True if a new cell
    is detected or False otherwise
2:  Let Num_Of_Cell_Passages ← Number of new cell passages detected so far
3:  Let [Xn,Yn] ← Location of the n'th cell passage
    (n ranges from 0 … Num_Of_Cell_Passages)
4:  Let [Xrobot,Yrobot] ← Current location of the robot
5:  Let Dist(n) ← Distance between [Xn,Yn] and [Xrobot,Yrobot]
    (n ranges from 0 … Num_Of_Cell_Passages)
6:  Let Detect_Num(n) ← The number of times that the n'th cell passage is
    detected during the current Search behavior.
    (n ranges from 0 … Num_Of_Cell_Passages)
7:  Let Cell_Passage_Size ← Cell passages size
8:  New_Cell_Detect ← True
9:  for i ← 0 to Num_Of_Cell_Passages do
10:     if Detect_Num(i) ≥ 2 then
11:         Delete all the cell passages information of the current cell
12:         Goto 16
13:     if Dist(i) < Cell_Passage_Size then
14:         New_Cell_Detect ← False
15:         Goto 16
16: if New_Cell_Detect = True then
17:     Save [Xrobot,Yrobot] as a new cell passage
18: return New_Cell_Detect
```

4.3 Communication

When dealing with multi robot systems the need for communication between robots often emerges. We are looking for type of communication methods that should be suitable for systems containing an arbitrary number of robots.

Our algorithm requires that each robot should have the ability to transmit Its ID and receive others, which means that we need to have some kind of way for robots to communicate, but the question we are facing is how to accomplish this with the common resources that today's robotic platforms have to offer?

We came to the conclusion that an IR (Infrared) communication will be the best solution for the following reasons:

- Many of today's robotic platforms have an IR remote control, meaning that IR capabilities already exist and no additional hardware is required.
- Our algorithm relies on the fact that a robot will communicate only with the other robots that are in the same cell, meaning that we do not want the ID transmission of a certain robot in certain cell to be picked up by another robot in another cell. Since IR energy can not break through walls and firm objects, we are actually promised that a robot transmitting its IR energy in one cell will not be detected by another robot working in some other cell, which is exactly what we wished for.

5 Experiments

To evaluate our algorithm I used a multi robot simulator that we specially developed for this purpose. When we test a certain algorithm we actually perform a sequence of tests where each one is done using different number of robots (starting from 1 robot and rising up to 10). Each test includes 50 runs, and each run is completed when either the robot covers over 95% of the work area or 90 simulated minutes elapse. The main goal of the tests is to confirm our assumption that when our algorithm is planted into any one of the tested algorithms the area coverage performance is improved by:

- Maximizing the area coverage:
  We show that when our algorithm is being used, robots cover more area in less time.
- Minimizing the standard deviation over multiple operations:
  We show that when we use a certain algorithm on a certain environment over and over again, the area coverage results are much more different from one operation to the other than when our algorithm is being used.

Minimizing the standard deviation is a very important goal, to understand why it is easier to look at the following example: let's take for example the vacuum cleaning application and assume that our task is to clean an entire hotel floor. If we offer the hotel a solution that will be satisfying only in 90% of the time then it is not good enough because this means that in the other 10% of the time the hotel guests might not be satisfied with the cleaning. We need the ability to provide some kind of consistency in the coverage time because many applications require consistent performance.

To test the hypotheses examined in the various experiments, we use a t-Test. We will use a one-tailed test when our hypothesis is that the mean of sample 1 is either higher or lower than the mean of sample 2 or a two-tailed test when our hypothesis is that the means of the two samples differ, no matter which one is higher and which is lower. We will consider the results to be statistically significant if the one-tailed probability is lower than 0.05, which is the conventional value.

5.1 Simulator

We developed a multi robot simulator using Visual C++ to support this research. The simulator models a 2-dimensional environment, supports the creation of any number of robots, supports the creation of any type of environment and obstacles, supports the common sensors our algorithm uses, provides the ability to enter random odometer errors during forward and turn movements and provides a variety of analysis information such as coverage percent, how many times the robot ran over the same spots and a simulated thermo map which shows if the robot spent more time in some areas than in other.

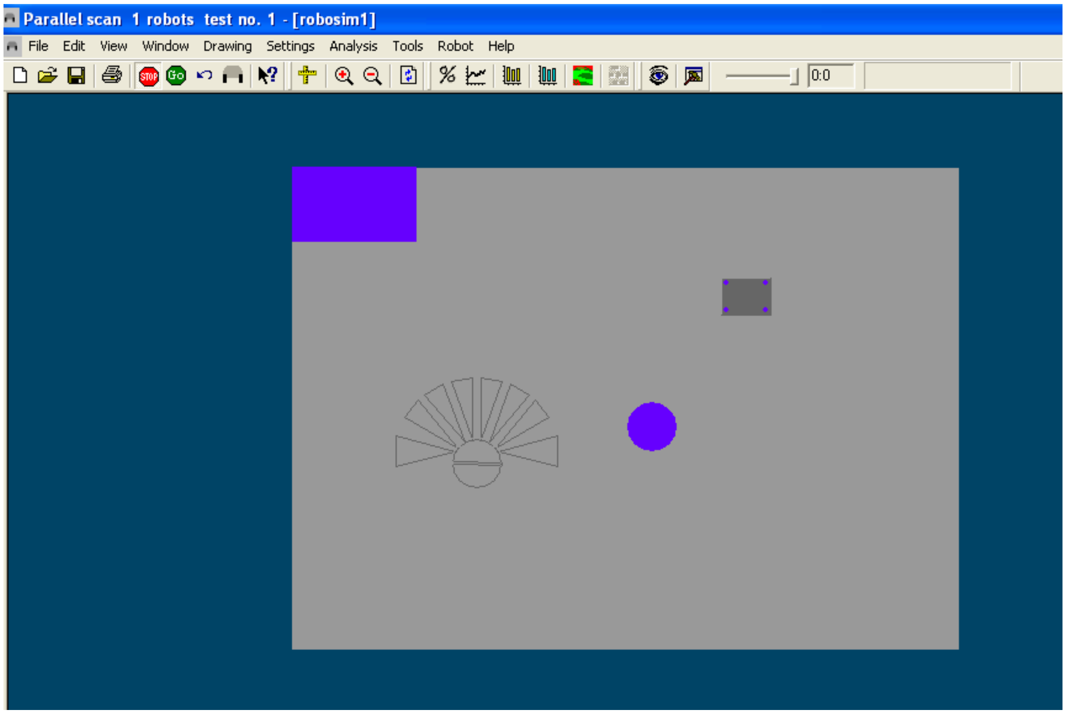


Figure 5.1: A screenshot of the simulator

The following figure (Figure 5.2) shows a test area we used in our experiments which actually simulates an office floor in size of 2500m².

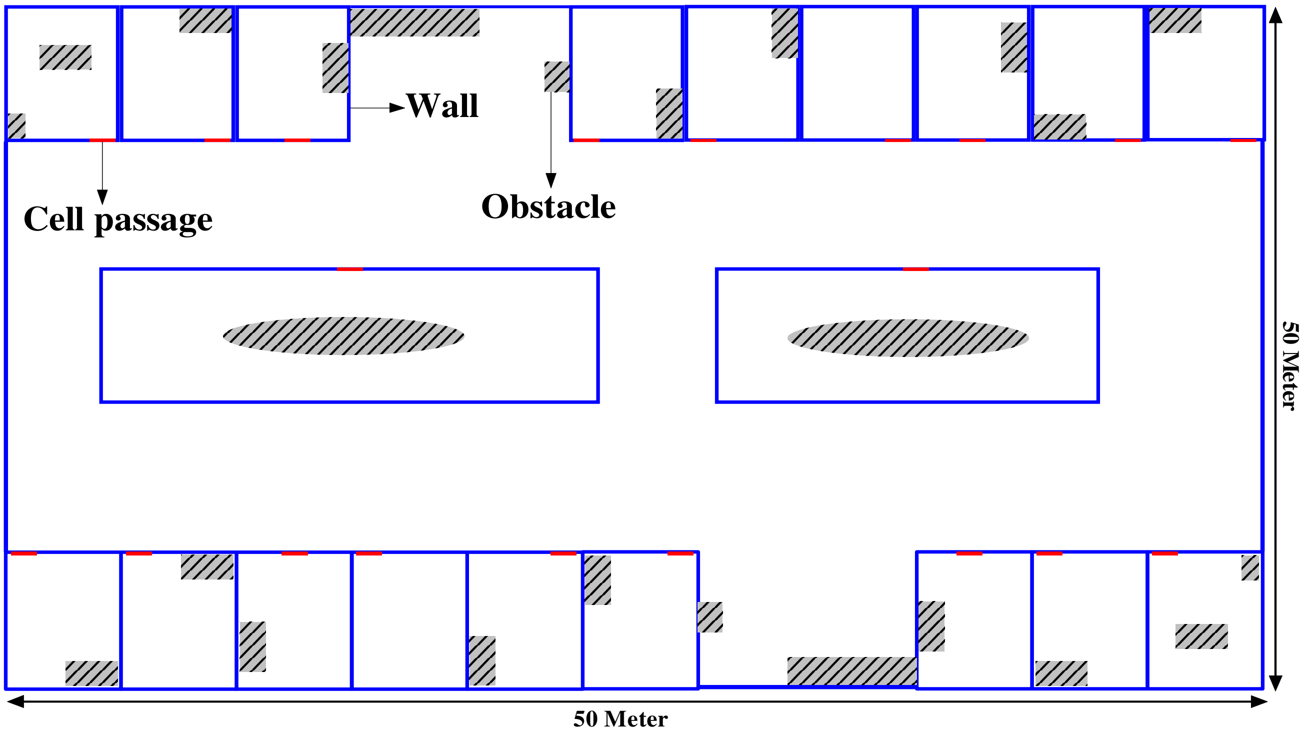


Figure 5.2

The simulator models the entire set of sensors our algorithm uses and allows the user to configure the location and range of the sensors. We will now describe the variety of sensors we use in our simulator which represent the common sensors which can be found in many of today's robotic platforms. To be more realistic we simulated a specific robotic platform called the FriendlyVac which is a robotic vacuum cleaner manufactured by Friendly Robotics [9].

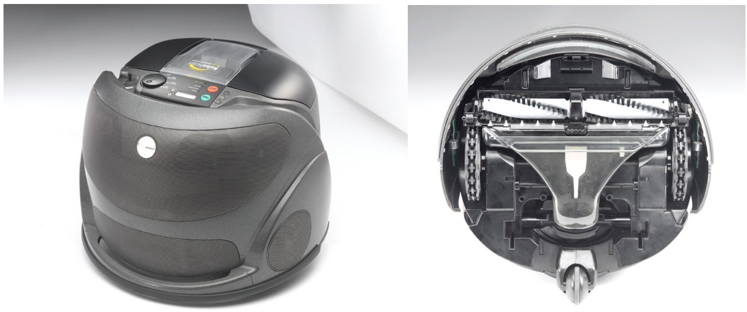


Figure 5.3: The FriendlyVac

Ultrasound sensors

There are 8 ultrasound sensors looking forward and 2 looking aside.

Stairs sensors

There are 2 IR sensors located in front of the drive wheels, facing down to detect stairs.

Bumper sensor

There are 2 analog sensors reading the bumper movements.

Virtual borders

The door sensor will give us the virtual borders capability and allow us to detect cell passages. It is actually an IR receiver and transmitter capable of detecting a special type of sticker which can be placed above doors. We can see that by using the current available robot's sensors we are able to achieve the ability to bound certain areas, turning them into cells and detecting them whenever we move from one cell to another.

IR sensors

There are 5 IR receivers capable of analyzing an IR transmission scattered around the robots body. These receivers will be used for detecting the transmission of other robots ID.

Odometers sensors

Odometer sensor for each drive wheel is used for measuring the speed and accumulated distance for each wheel.

The following figure (Figure 5.4) shows an overview of the entire FriendlyVac sensors deployment

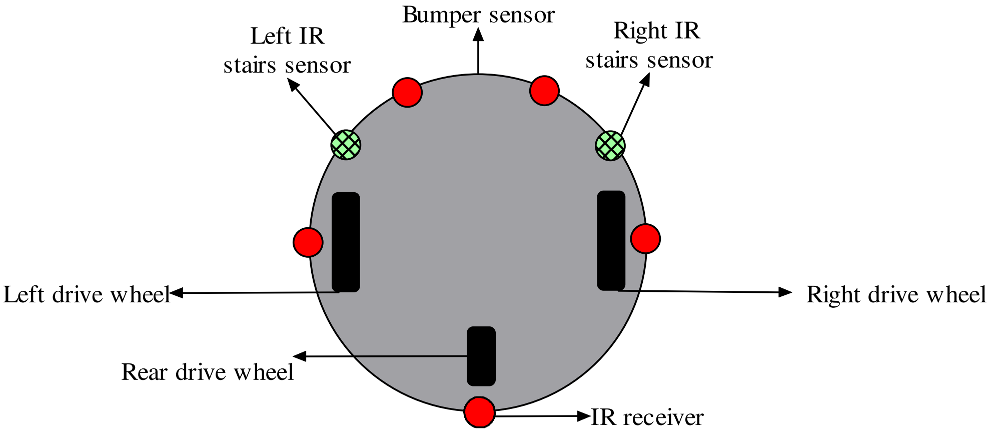


Figure 5.4

The hardware modifications that should be made to the FriendlyVac platform are minor. An IR transmitter with the ability to transmit n-Bit words (to support 2^n number of robots) should be assembled on the robot and wired into the MCU (Microcontroller unit) to give the robot the ability to transmit ID's ranging from 0 to 2^n.

We would like to note that the FriendlyVac, like many robotic platforms, comes with an IR remote control which can be used for the ID transmission and save the expense of another IR transmitter.

5.2 Comparative evaluation of coverage algorithms

Here we present the results received from comparative evaluation we performed on the parallel, random and spiral coverage algorithms. The tests are performed on the office floor map (Figure 5.2).

The following figure (Figure 5.5) shows the area coverage results.

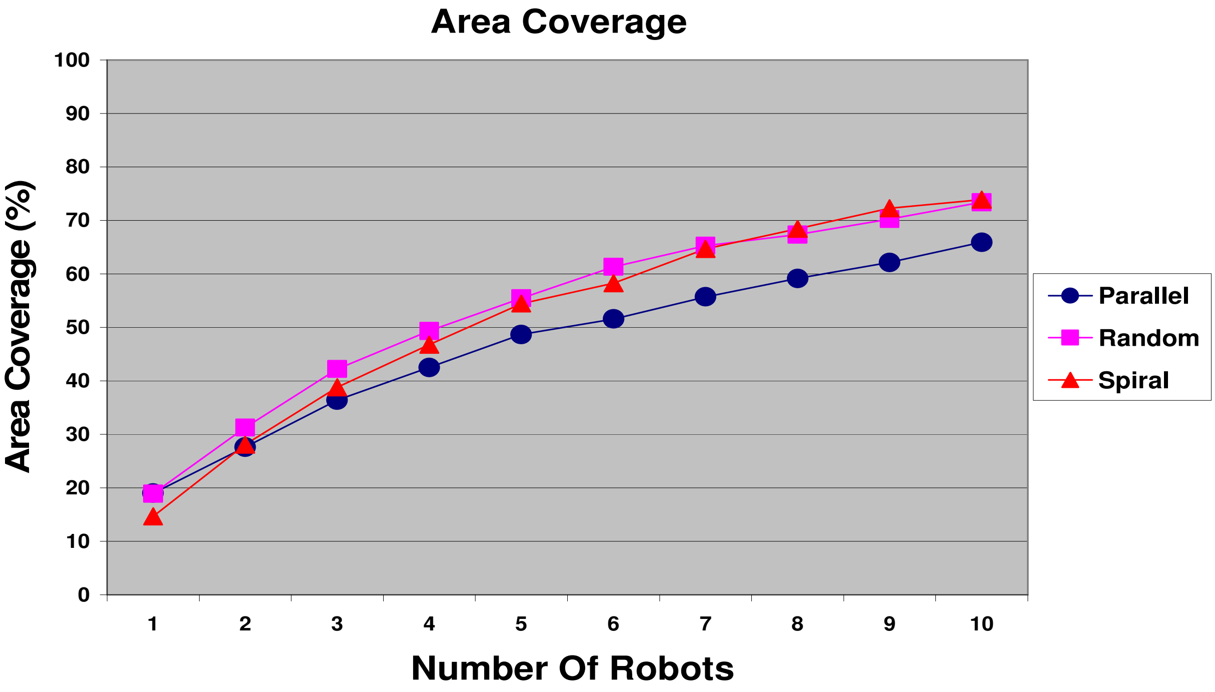


Figure 5.5

A two-tailed t-test between the random and the spiral area coverage results shows only a moderated statistical significant indication that one of them is better than the other (the result is 0.06).

A one-tailed t-Test between the random and the parallel area coverage results shows that there is a statistical significant indication that the random is better than the parallel (the result is 2.94E-05).

A one-tailed t-test between the spiral and the parallel area coverage results shows that there is a statistical significant indication that the spiral is better than the parallel (the result is 0.006).

The following figure (Figure 5.6) shows the standard deviation results.

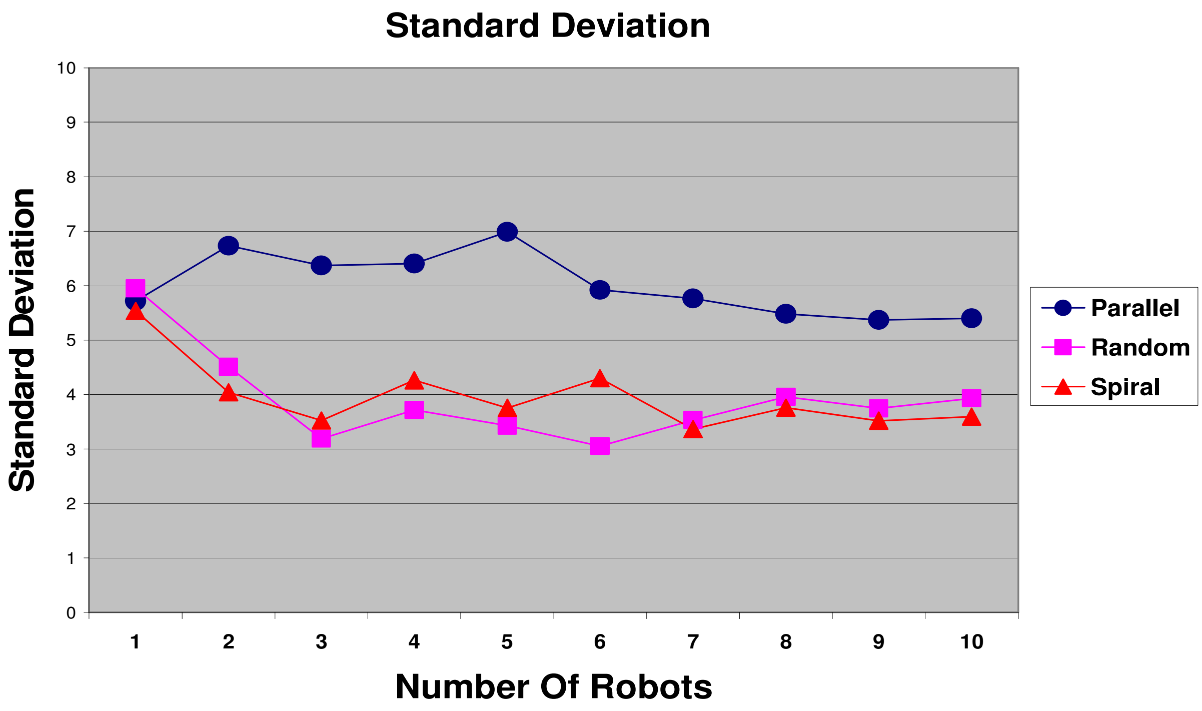


Figure 5.6

A two-tailed t-test between the random and the spiral standard deviation results shows that there is no statistical significant indication that one of them is better than the other (the result is 0.72).

A one-tailed t-test between the random and the parallel standard deviation results shows that there is a statistical significant indication that the random is better than the parallel (the result is 8.46E-05).

A one-tailed t-test between the spiral and the parallel standard deviation results shows that there is a statistical significant indication that the spiral is better than the parallel (the result is 1.56E-05).

5.3 Estimation formulas

Our algorithm uses different estimation formulas (Section 4.2.2) such as the cell size estimation and the required work time estimation. In order to confirm our estimation formulas accuracy we performed a sequence of tests on various types of cells. We will now present the results for one of the tested cells (Figure 5.7) which is actually a square shape with size of 100m².

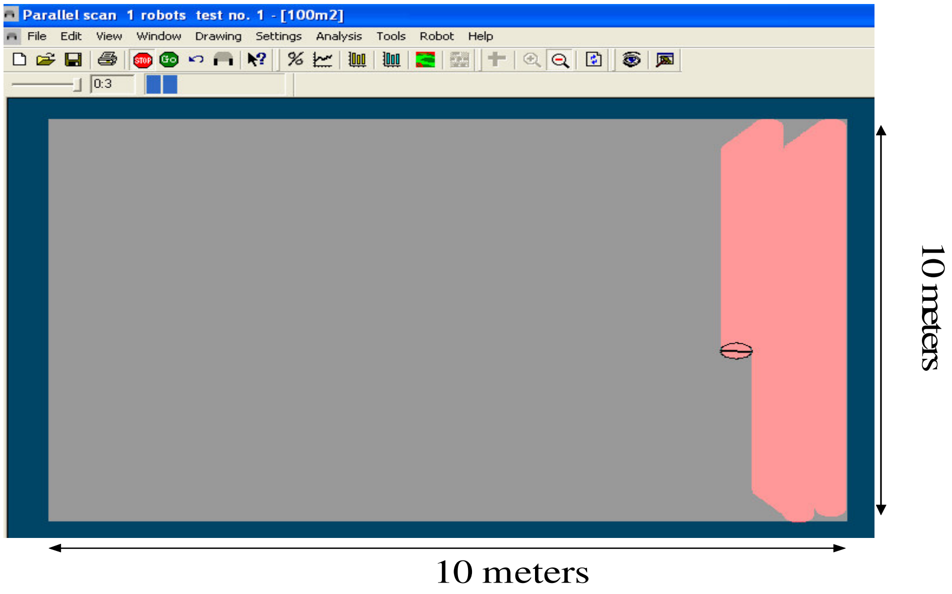


Figure 5.7

We performed 100 simulation runs on this cell and received the following results:

- The average of the estimated cell size is 98.4m^2 and the standard deviation is 1.1
- Using our required work time estimation we receive the following area coverage results: The average of the coverage percent is 98.58% and the standard deviation is 1.07

Our experiments show that the received estimation formulas accuracy is good enough.

5.4 Our algorithm

The section is a continuation to the comparative evaluation of coverage algorithms we performed before because here we add our algorithm to each one of these three algorithms and show that an improvement has been achieved in the coverage performance for each one of the algorithms.

The following figure (Figure 5.8) shows the area coverage results for the parallel algorithm.

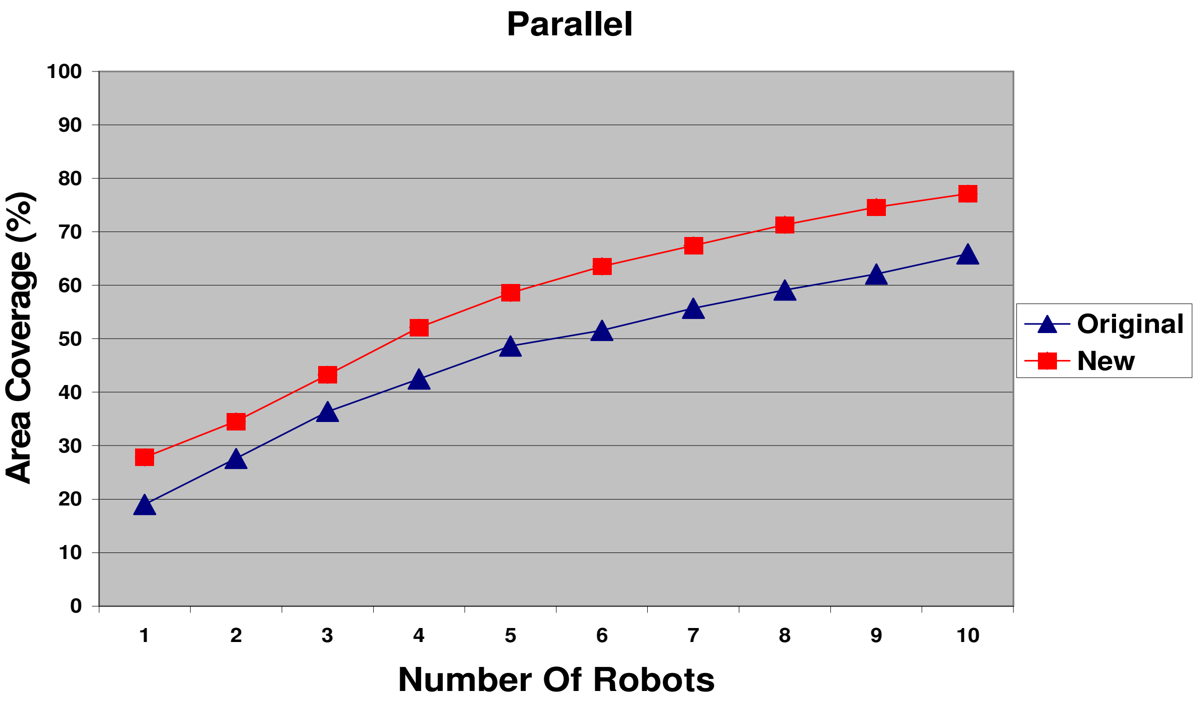


Figure 5.8

A one-tailed t-test performed on the area coverage results for the parallel algorithm shows there is a statistical significant indication that our algorithm improves the area coverage (the result is 4.64E-08).

The following figure (Figure 5.9) shows the standard deviation results for the parallel algorithm.

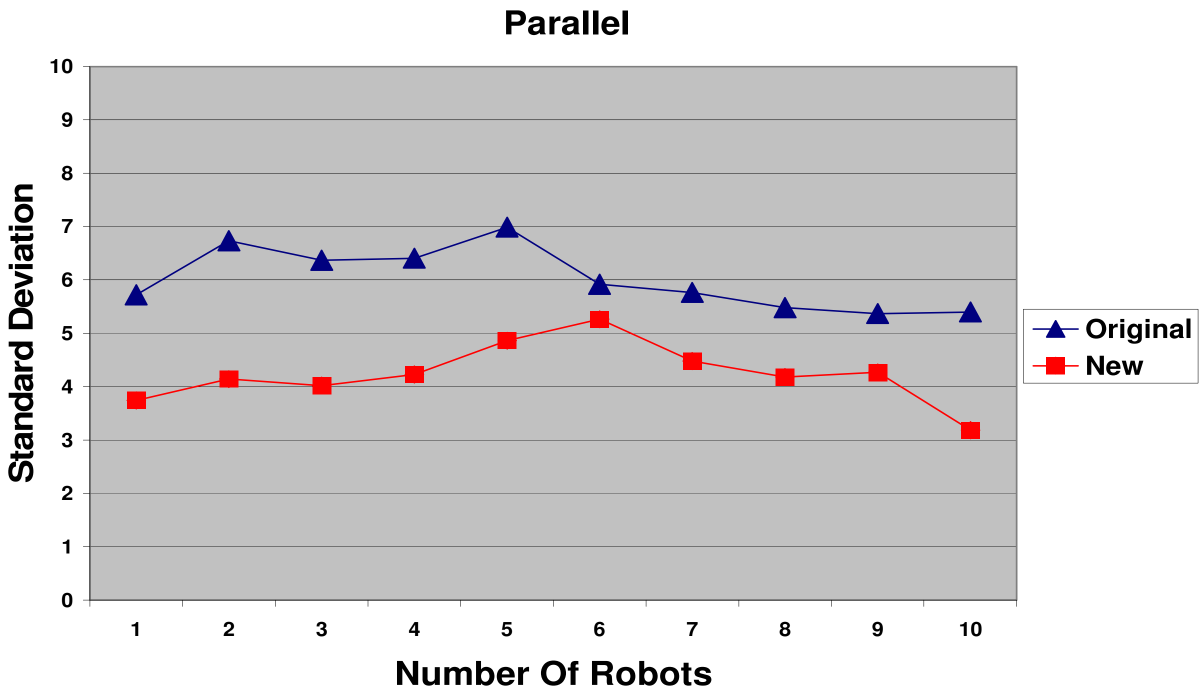


Figure 5.9

A one-tailed t-test performed on the standard deviation results for the parallel algorithm shows there is a statistical significant indication that our algorithm improves the standard deviation (the result is 5.19E-06).

The following figure (Figure 5.10) shows the area coverage results for the random algorithm.

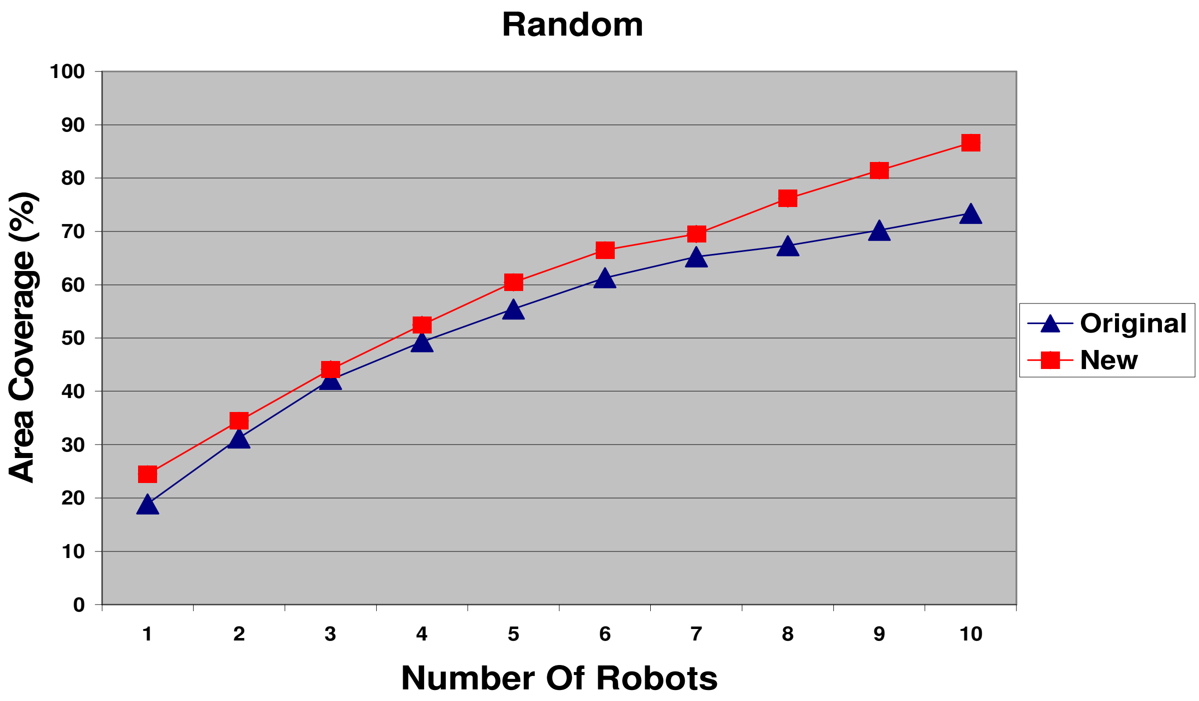


Figure 5.10

A one-tailed t-test performed on the area coverage results for the random algorithm shows there is a statistical significant indication that our algorithm improves the area coverage (the result is 0.0002).

The following figure (Figure 5.11) shows the standard deviation results for the random algorithm.

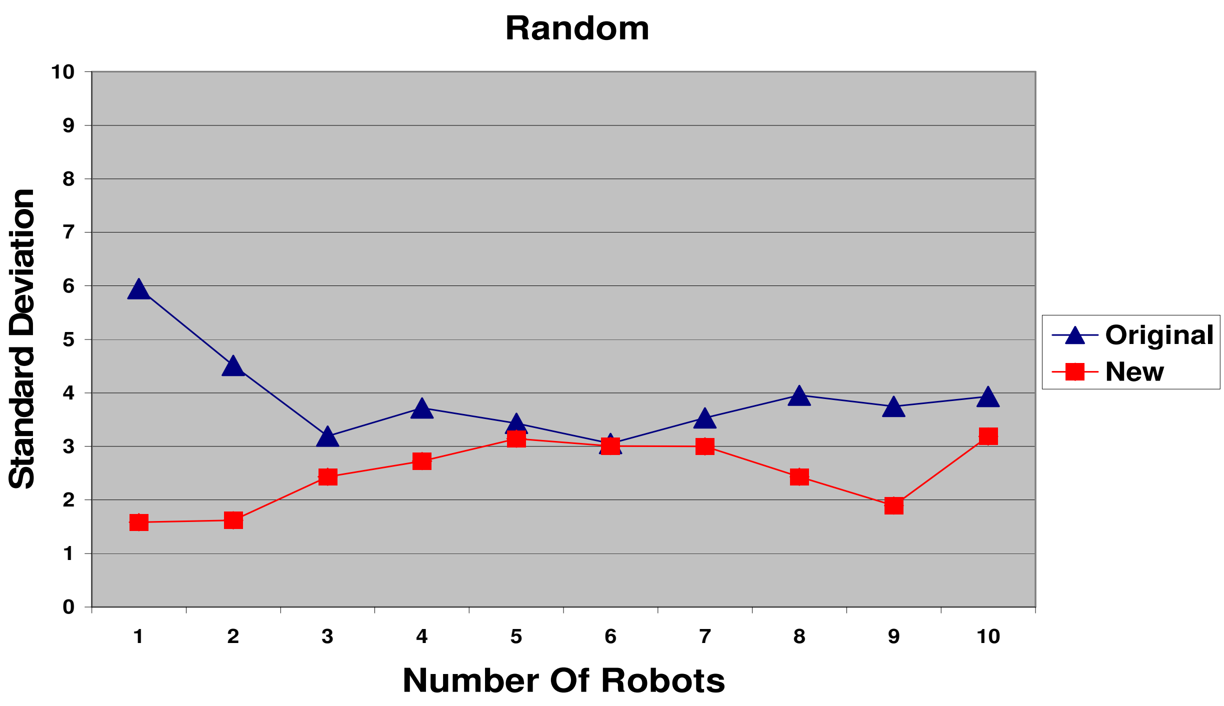


Figure 5.11

A one-tailed t-test performed on the standard deviation results for the random algorithm shows there is a statistical significant indication that our algorithm improves the standard deviation (the result is 0.004).

The following figure (Figure 5.12) shows the area coverage results for the spiral algorithm.

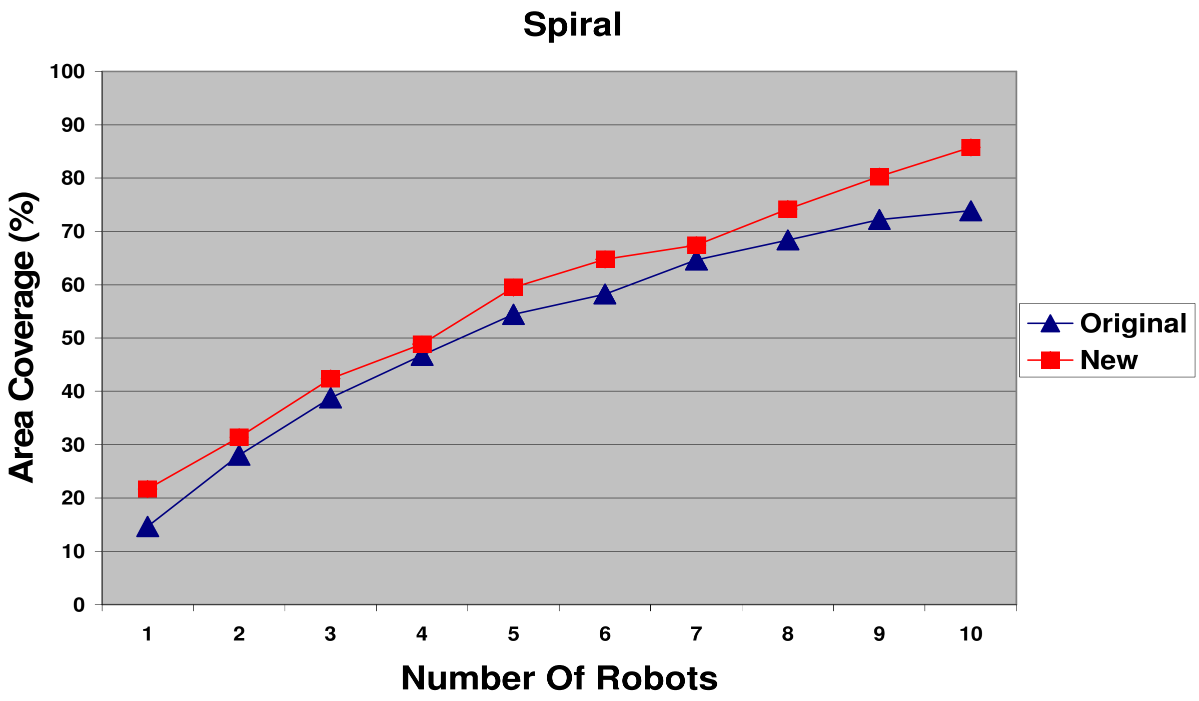


Figure 5.12

A one-tailed t-test performed on the area coverage results for the spiral algorithm shows there is a statistical significant indication that our algorithm improves the area coverage (the result is 9.72E-05).

The following figure (Figure 5.13) shows the standard deviation results for the spiral algorithm.

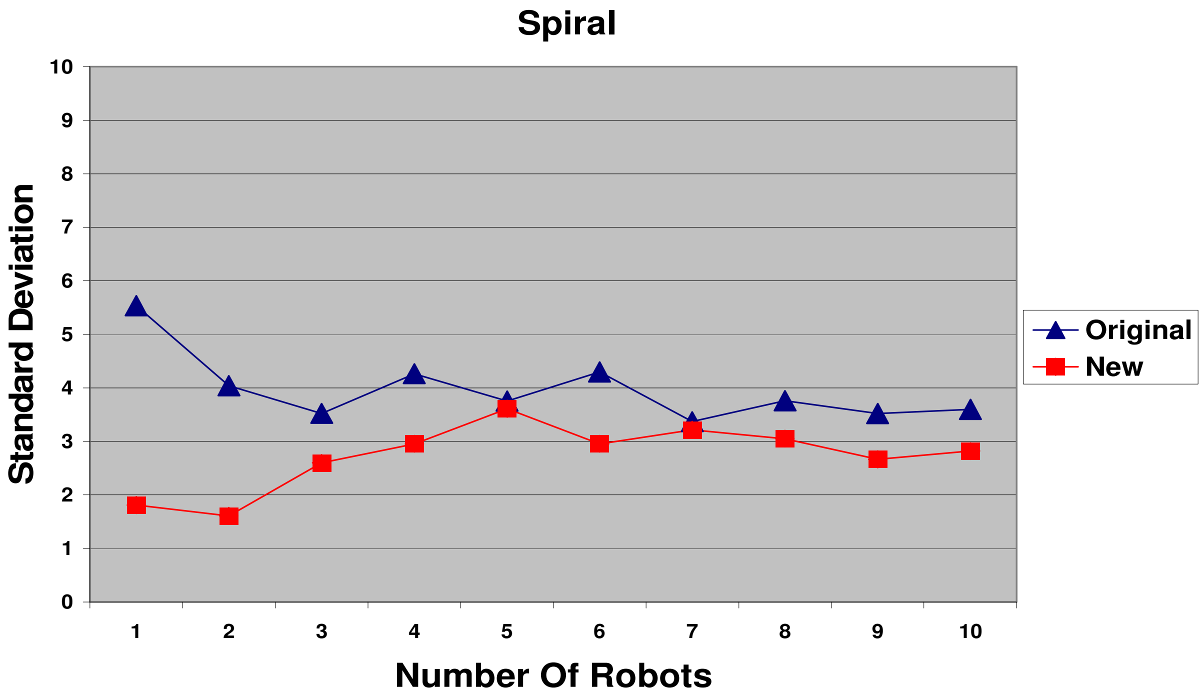


Figure 5.13

A one-tailed t-test performed on the standard deviation results for the spiral algorithm shows there is a statistical significant indication that our algorithm improves the standard deviation (the result is 0.002).

As we can see the results are very satisfying and they verify our assumptions about our algorithm since they show that there is a statistically significant improvement in both the area coverage and the standard deviation of all the algorithms we tested.

5.5 Obstacles influence

In this section we would like to test how obstacles influence our algorithm. We intend to test the office floor with obstacles scattered on 5%, 10% and 20% of the work area.

The following figure (Figure 5.14) shows an example of how the 5% obstacles are scattered in the work area. The small circles are the obstacles.

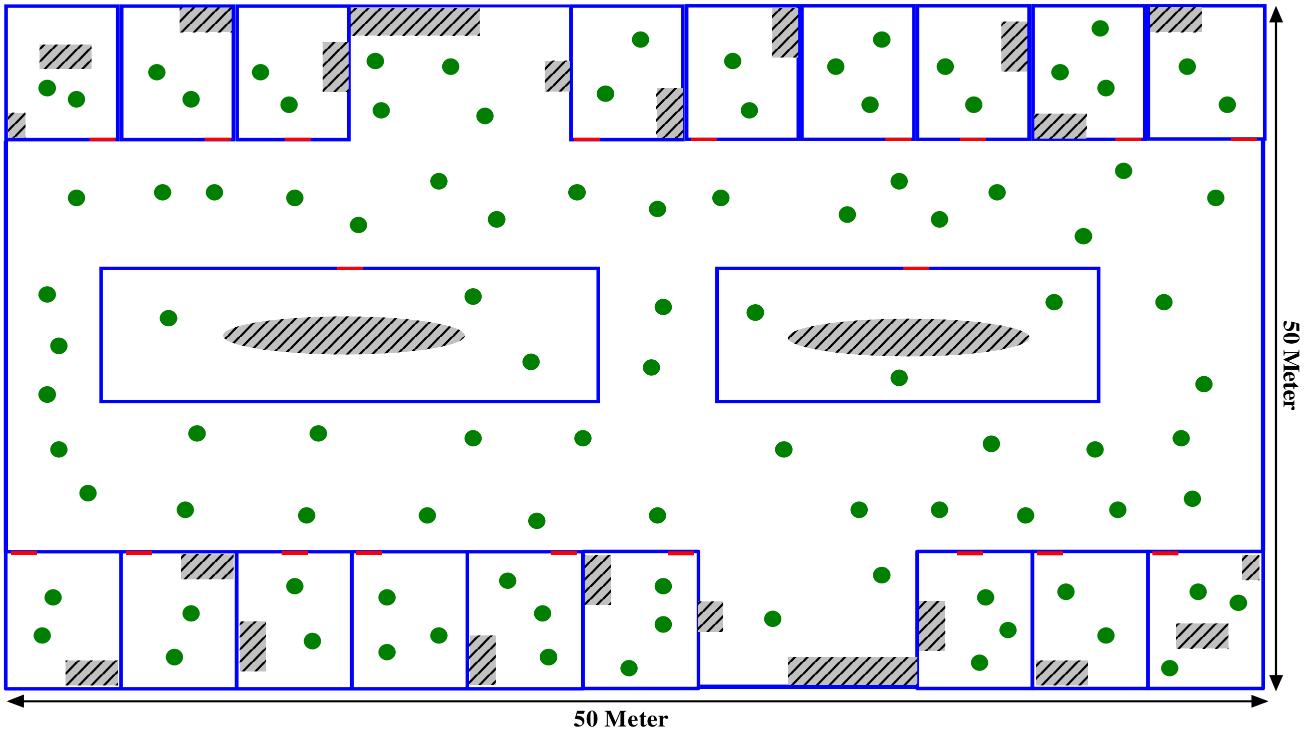


Figure 5.14

The following figure (figure 5.15) shows the area coverage results for the parallel algorithm.

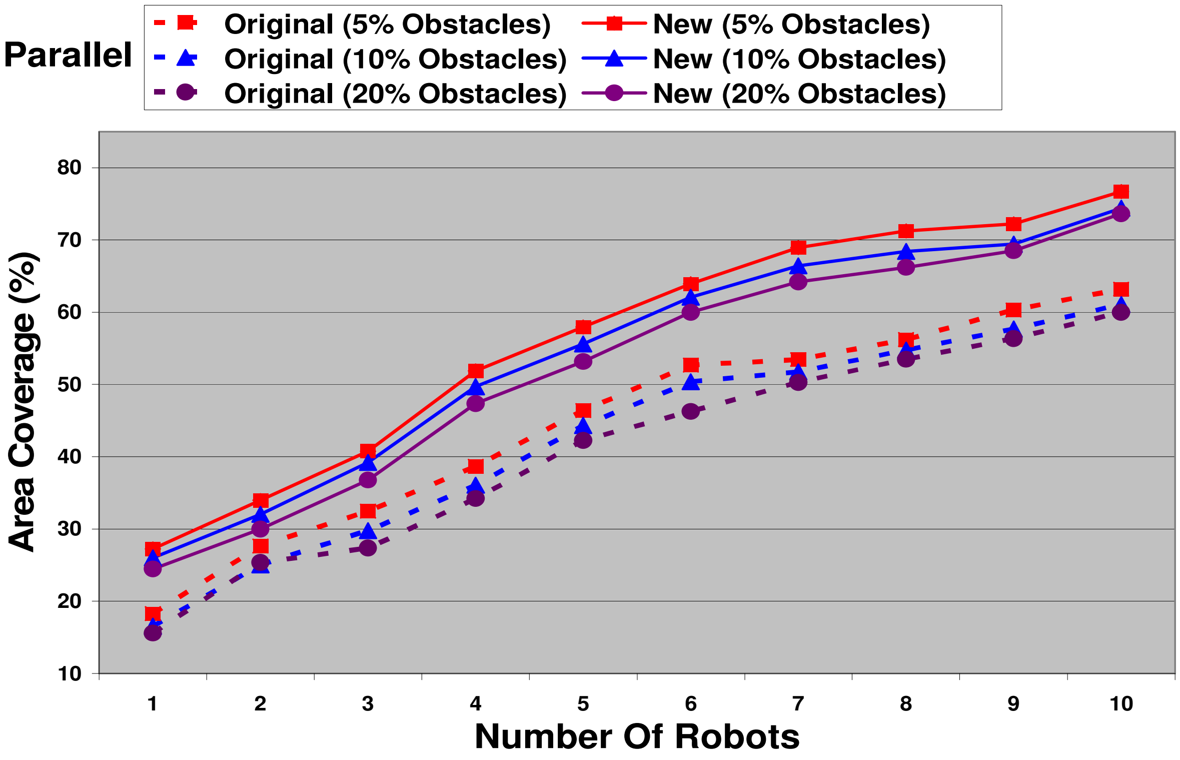


Figure 5.15

A one-tailed t-test performed on the area coverage results for the parallel algorithm shows there is a statistical significant indication that our algorithm improves the area coverage in the 5% (the result is 3.03E-07), 10% (the result is 4.70E-08) and 20% (the result is 3.54E-07) obstacles tests.

The following figure (Figure 5.16) shows the standard deviation results for the parallel algorithm.

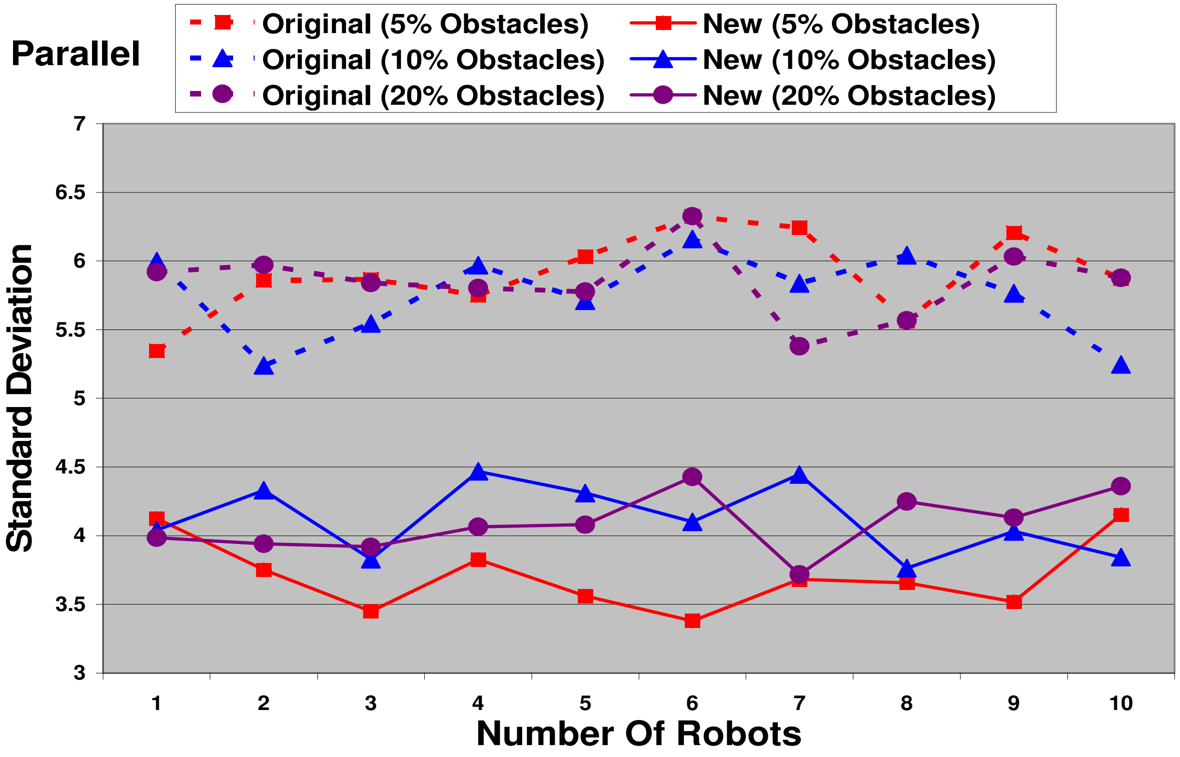


Figure 5.16

A one-tailed t-test performed on the standard deviation results for the parallel algorithm shows there is a statistical significant indication that our algorithm improves the standard deviation in the 5% (the result is 1.45E-07), 10% (the result is 1.98E-07) and 20% (the result is 5.60E-10) obstacles tests.

The following figure (figure 5.17) shows the area coverage results for the random algorithm.

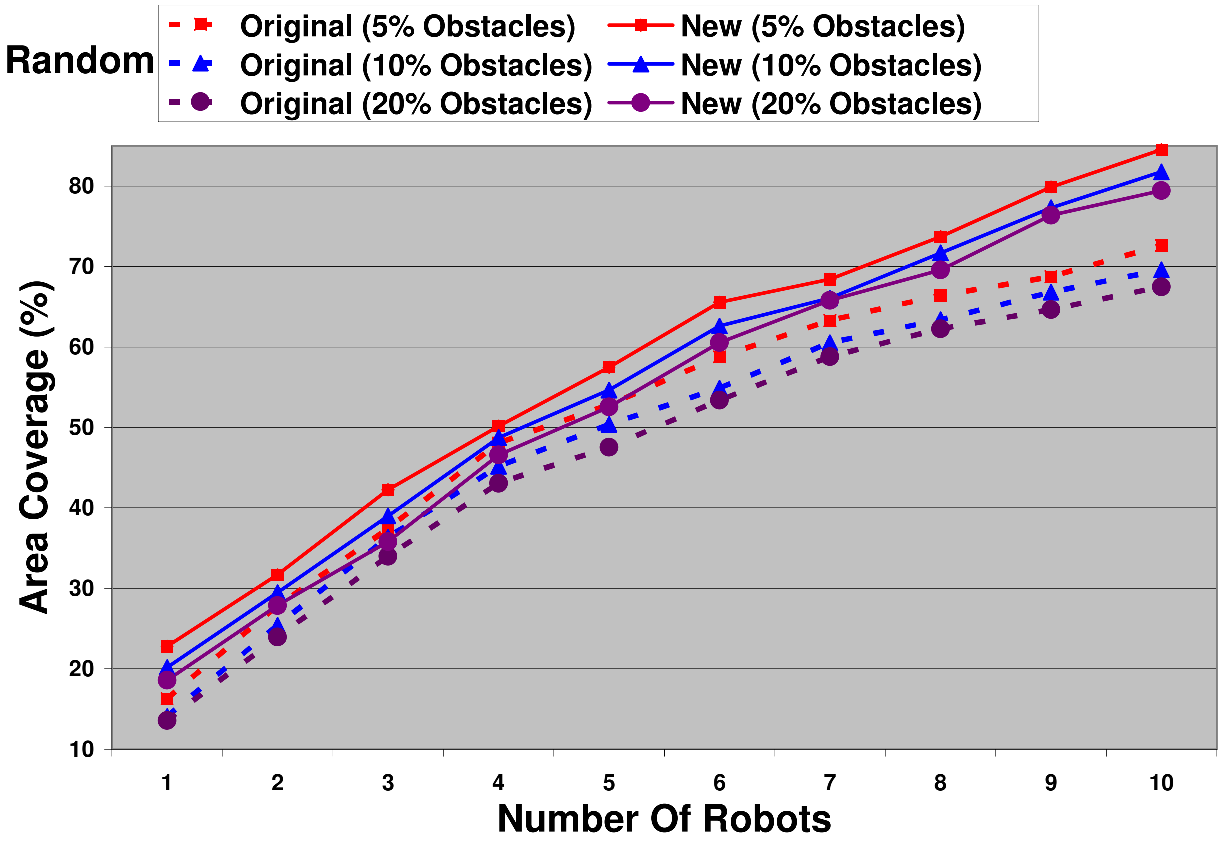


Figure 5.17

A one-tailed t-test performed on the area coverage results for the random algorithm shows there is a statistical significant indication that our algorithm improves the area coverage in the 5% (the result is 5.02E-05), 10% (the result is 5.33E-05) and 20% (the result is 8.97E-05) obstacles tests.

The following figure (Figure 5.18) shows the standard deviation results for the random algorithm.

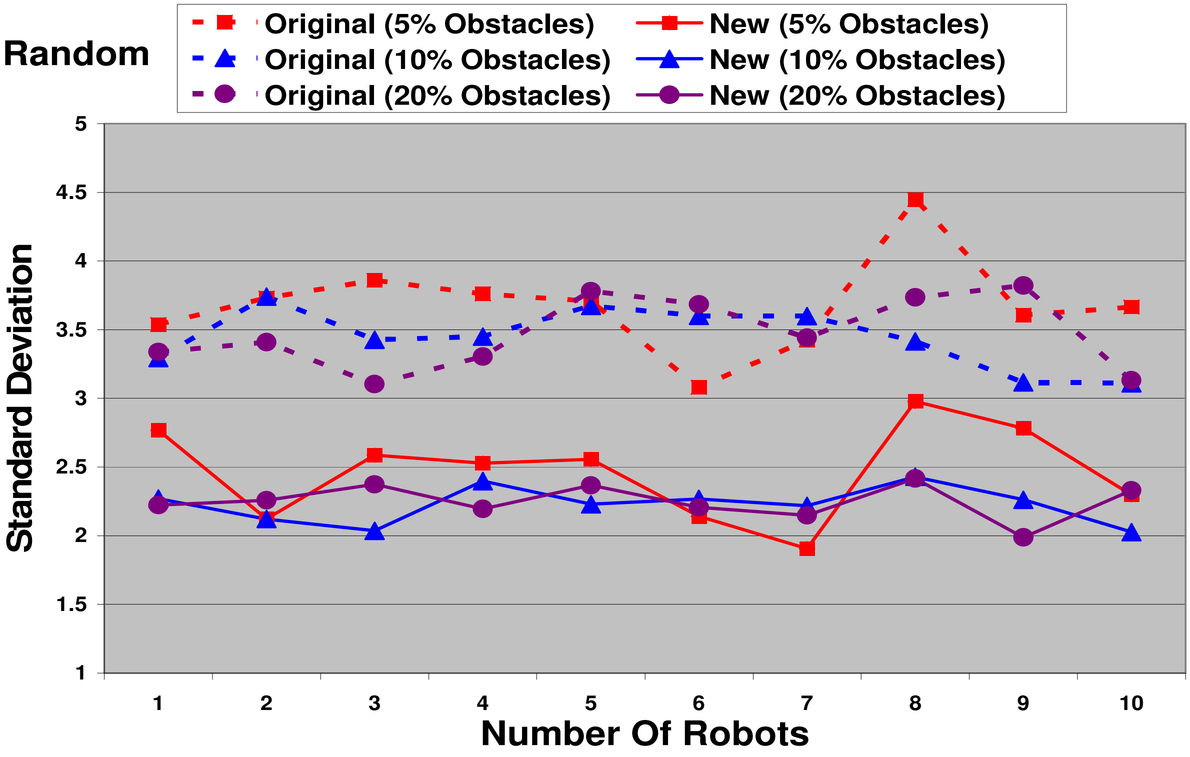


Figure 5.18

A one-tailed t-test performed on the standard deviation results for the random algorithm shows there is a statistical significant indication that our algorithm improves the standard deviation in the 5% (the result is 1.76E-07), 10% (the result is 4.09E-08) and 20% (the result is 3.90E-07) obstacles tests.

The following figure (figure 5.19) shows the area coverage results for the spiral algorithm.

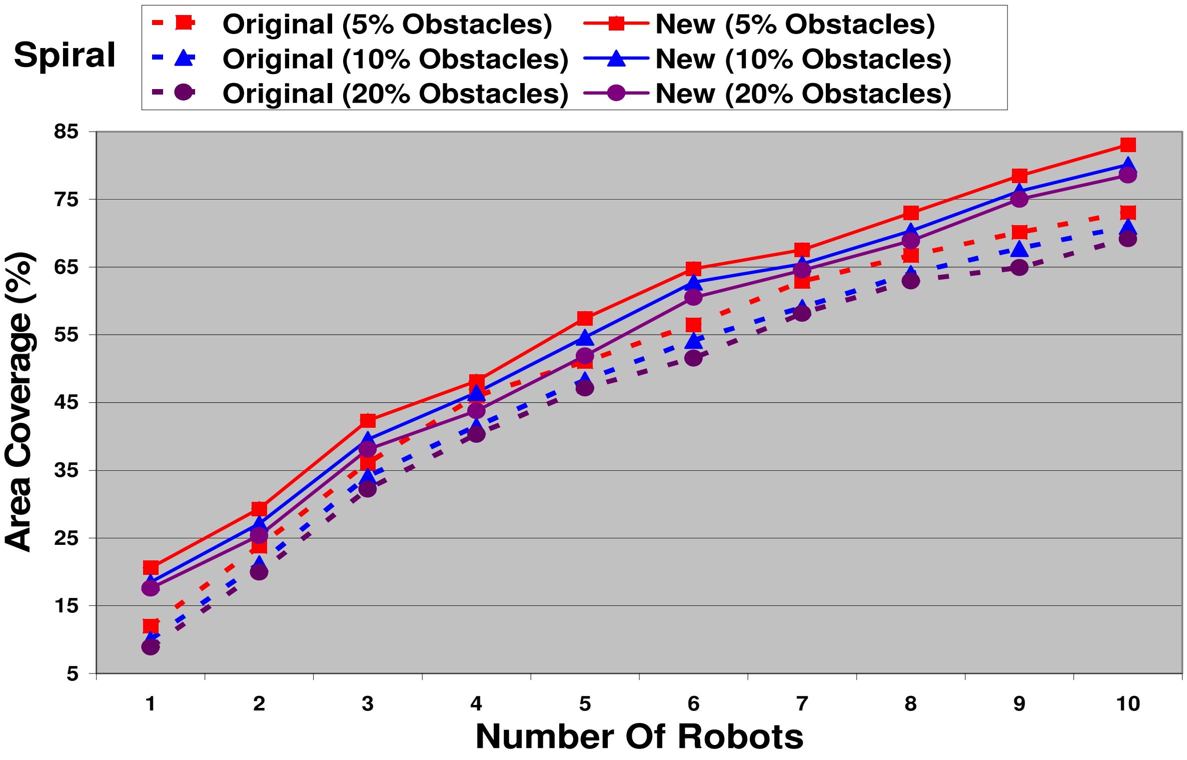


Figure 5.19

A one-tailed t-test performed on the area coverage results for the spiral algorithm shows there is a statistical significant indication that our algorithm improves the area coverage in the 5% (the result is 3.16E-06), 10% (the result is 6.59E-08) and 20% (the result is 2.11E-06) obstacles tests.

The following figure (Figure 5.20) shows the standard deviation results for the spiral algorithm.

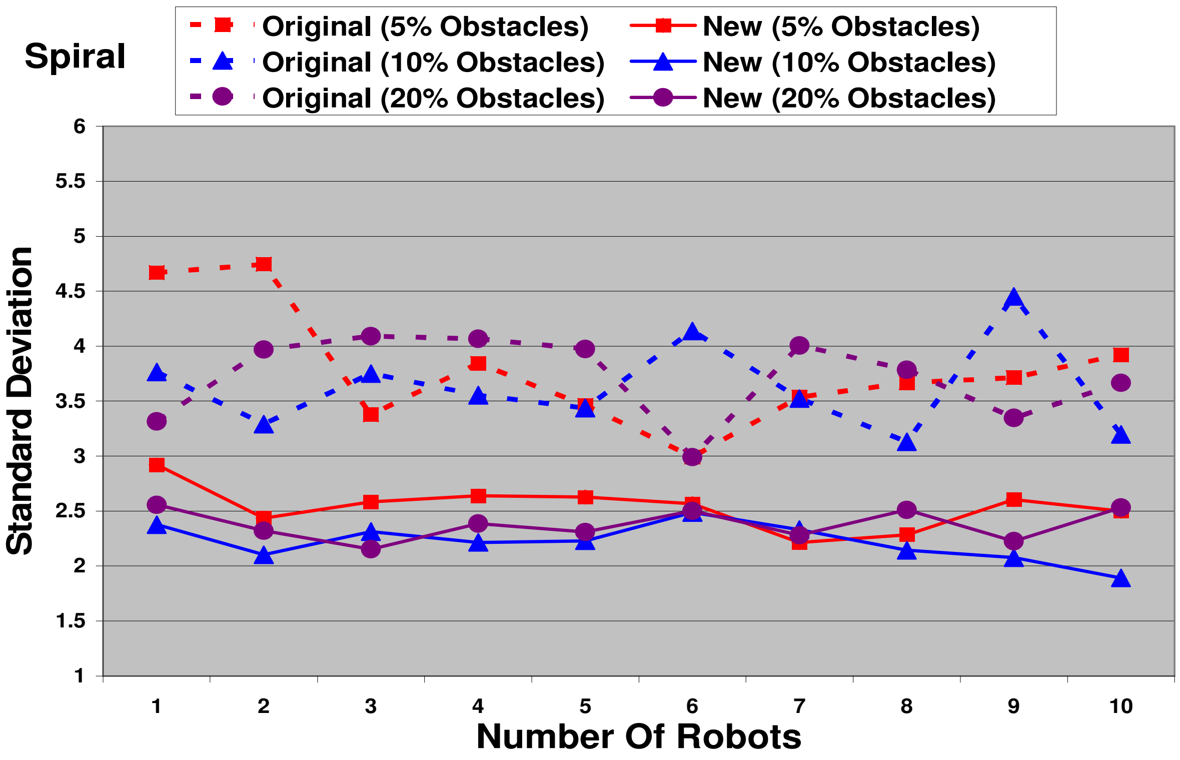


Figure 5.20

A one-tailed t-test performed on the standard deviation results for the spiral algorithm shows there is a statistical significant indication that our algorithm improves the standard deviation in the 5% (the result is 1.79E-05), 10% (the result is 5.17E-07) and 20% (the result is 4.03E-06) obstacles tests.

As we can see our algorithm is not affected by multiple obstacles scattering since it maintained its better results over the original algorithms in both the area coverage and the standard deviation.

5.6 Cells free work areas

In this section we would like to test how work areas with no cells influence our algorithm. This test should be interesting since when there are no cells our algorithms lose its advantages over the original ones. Using our simulator we created a cell free work area with size of 2500m^2.

The following figure (Figure 5.21) shows the cells free work area:

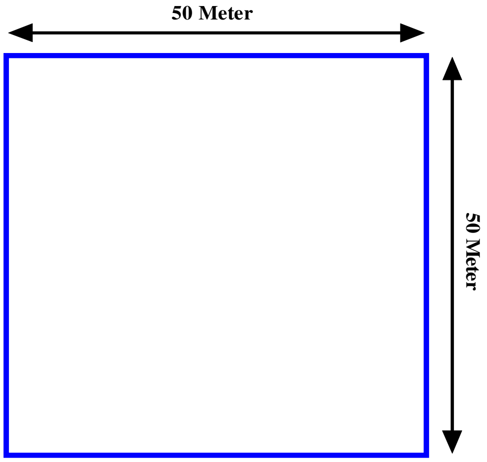


Figure 5.21: Cell free work area

The following figure (Figure 5.22) shows the area coverage results for all algorithms.

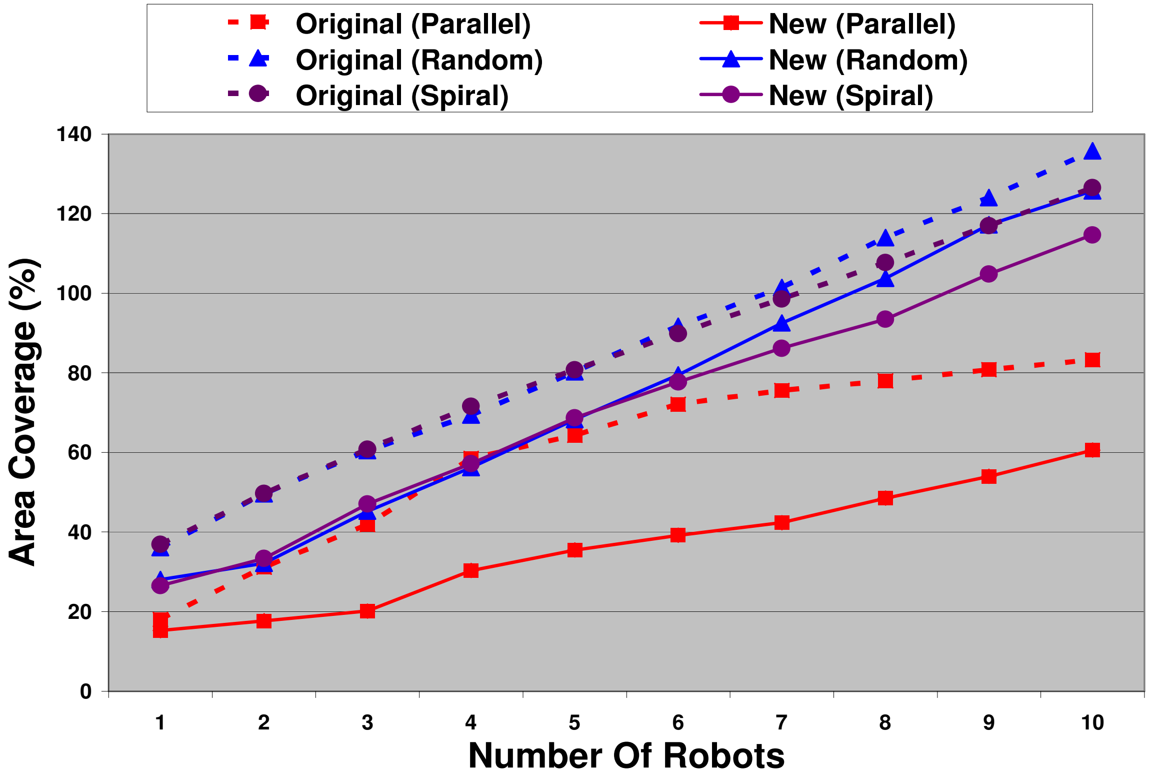


Figure 5.22

A one-tailed t-test performed on the area coverage results shows that there is a statistical significant indication that our algorithm decreases the area coverage in the parallel (the result is 1.08E-05), random (the result is 8.29E-07) and spiral (the result is 2.27E-09) algorithms.

The following figure (Figure 5.23) shows the standard deviation results for all algorithms.

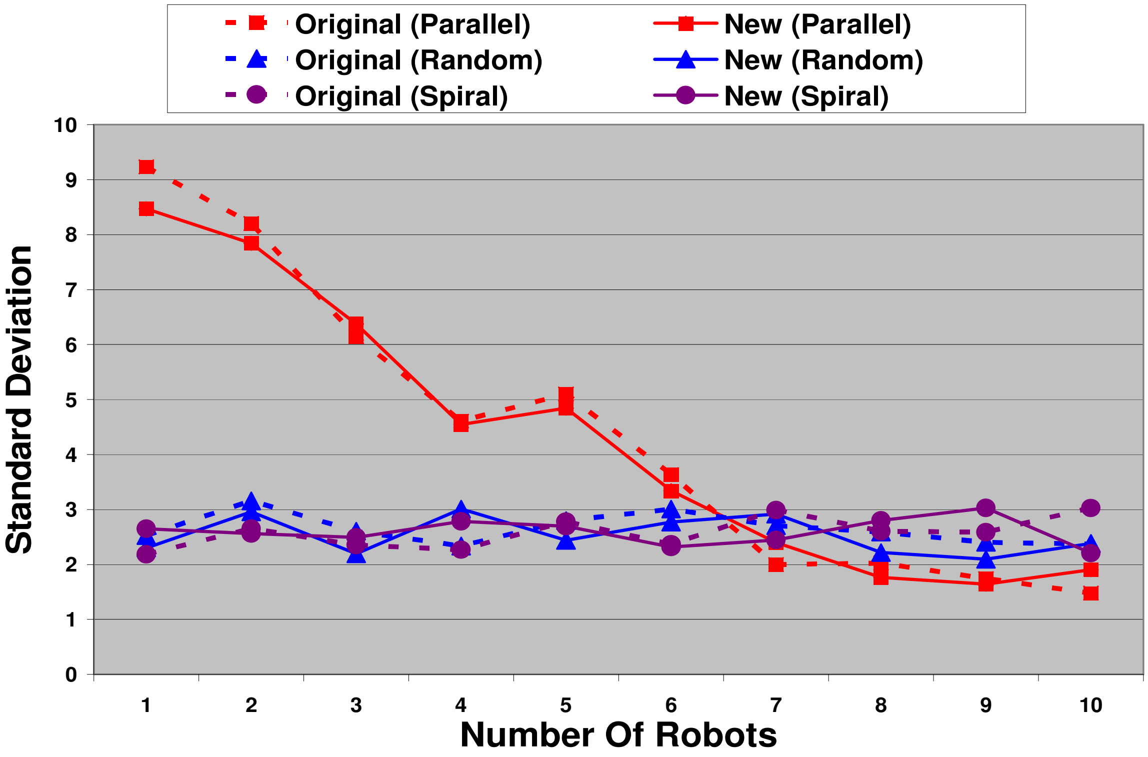


Figure 5.23

A one-tailed t-test performed on the standard deviation results shows that there is no statistical significant indication that our algorithm influences the standard deviation in the parallel (the result is 0.19), random (the result is 0.14) and spiral (the result is 0.45) algorithms.

The results are quite interesting. There is no statistical significant indication that our algorithm influences the standard deviation which is what we expected since when we operate in an area with no cells, our algorithm looses its advantages over the original ones and there is no reason for the standard deviation results to be better. But the area coverage results shows a statistical significant indication that our algorithm damages the area coverage. This can be explained by the fact that our algorithm spends valuable time in the Search behavior looking for cell passages that do not exist.

In order to solve this issue we decided to improve the Search behavior in the following way: If we do not detect any cell passages during the Search behavior it means we are working in an area with no cells and therefore we should disable the Search behavior until a cell passage is detected during the Scan behavior, we should also stop transmitting the ID (This change is farther explained in section 4.2.4). We will now repeat the cells free work area tests using our new improved algorithm in order to see the influence of the change we made.

The following figure (Figure 5.24) shows the area coverage results for all algorithms.

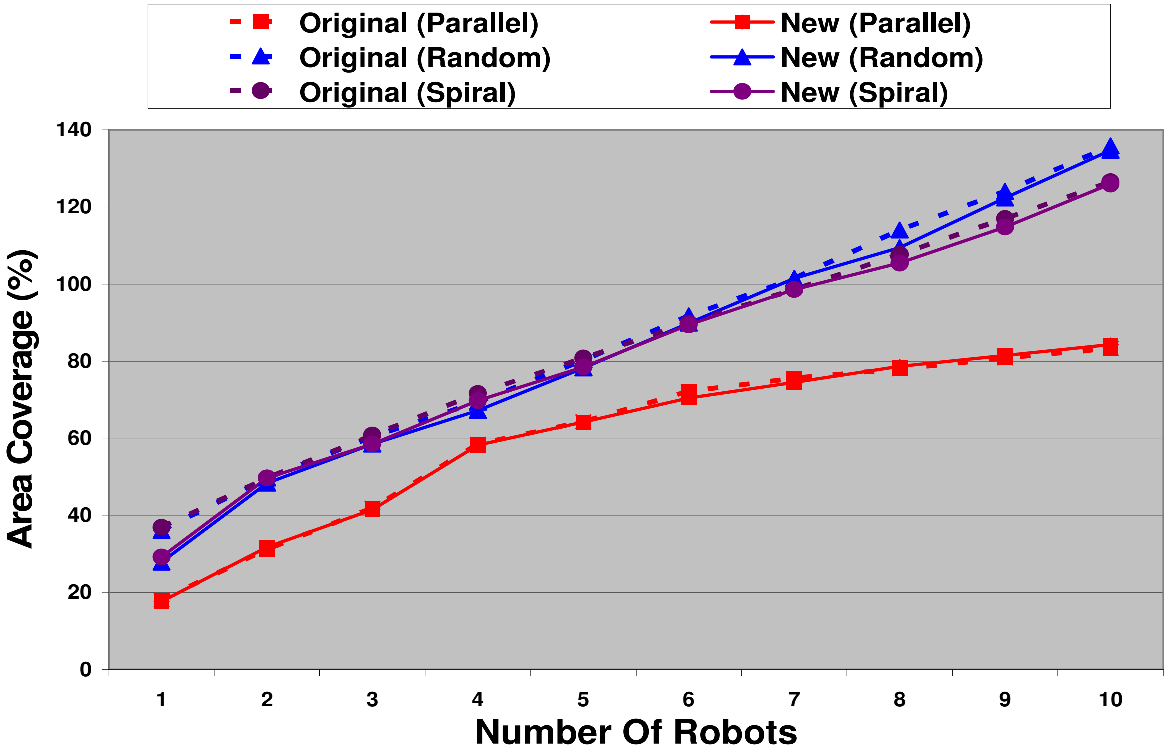


Figure 5.24

A one-tailed t-test performed on the area coverage results shows that there is a statistical significant indication that the improvement we made to our algorithm for supporting work areas without cells improves the area coverage in the parallel (the result is 9.75E-06), random (the result is 8.30E-05) and spiral (the result is 1.54E-06) algorithms.

The following figure (Figure 5.25) shows the standard deviation results for all algorithms.

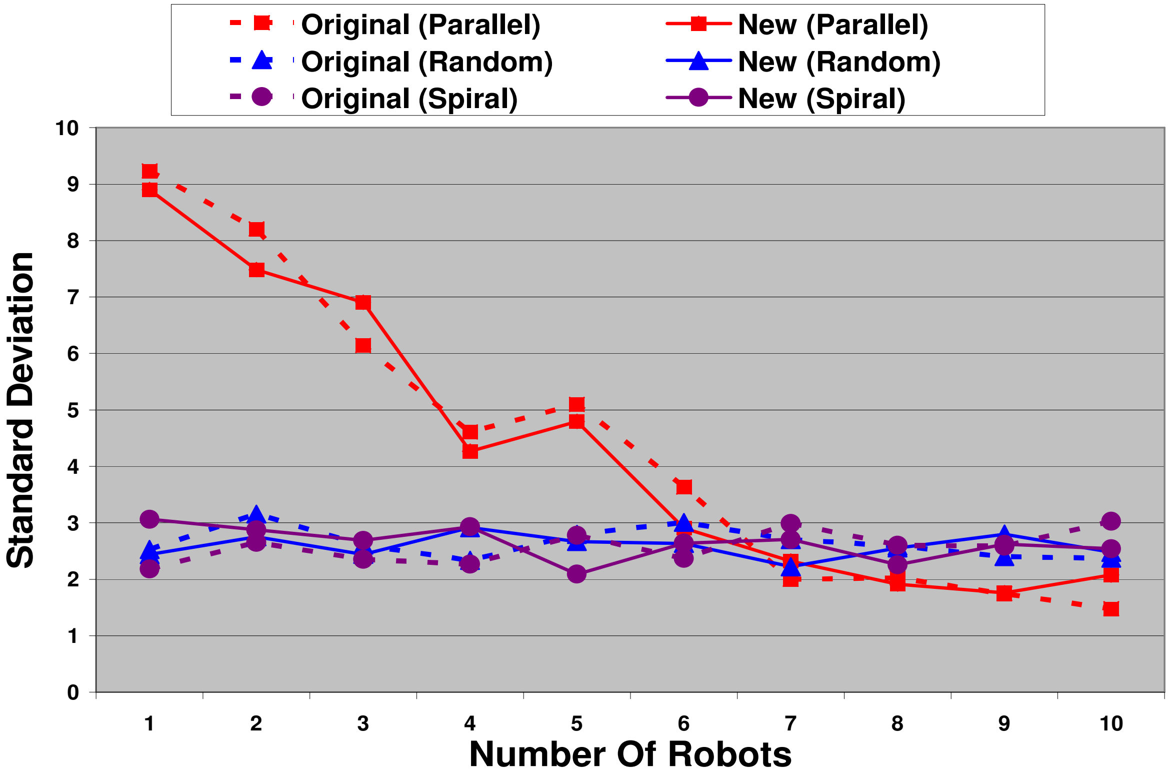


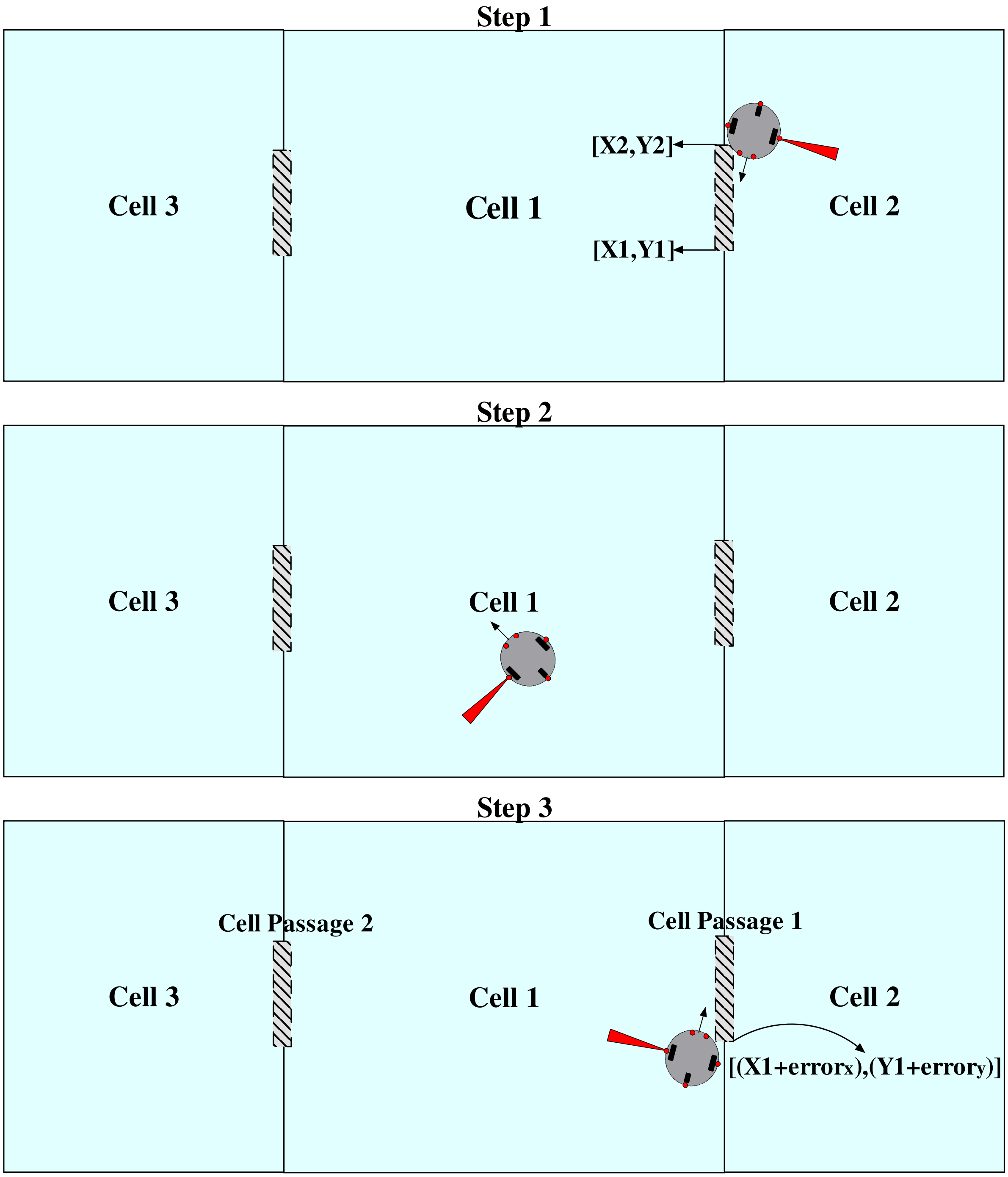
Figure 5.25

A one-tailed t-test performed on the standard deviation results shows that there is no statistical significant indication that our algorithm influences the standard deviation in the parallel (the result is 0.30), random (the result is 0.29) and spiral (the result is 0.35) algorithms, which is what we wanted to confirm here in order to make sure that the improvement we made to our algorithm for supporting work areas without cells didn't damage the standard deviation results in any way.

5.7 Odometry errors

Our algorithm relies on marking the locations of cell passages in order to enable robots to decide during the Search behavior if a cell passage should be passed through or skipped. Marking the location can be achieved in different ways and it will be up to whoever is using our algorithm to use the available capabilities of the robotic platform he has in hand in order to get the best location accuracy that can be achieved. In our algorithm we used the robot odometers to create a 2 dimensional grid for saving the [*X,Y*] location of cell passages. Monitoring the robot's location by using the odometers is a common thing in robotic platforms and it usually does the job pretty nicely, but it has its weaknesses. For example if the drive wheels slips from time to time due to bad surface grip then accumulated odometer errors can result bad cell passages location calculations, meaning that if a robots cell passage location is saved at a certain time as $[X,Y]$ then when the robot will reach this same cell passage after accumulating some odometer errors this same location will be $[(X + error_X),(Y + error_Y)]$ where the size of $error_X$ and $error_Y$ depends on the odometry accuracy. To understand the meaning of odometer errors and the influence they can have on our algorithm we should look at the following example.

The following figure (figure 5.26) illustrates how odometer errors can damage our algorithm performance.



Step 4

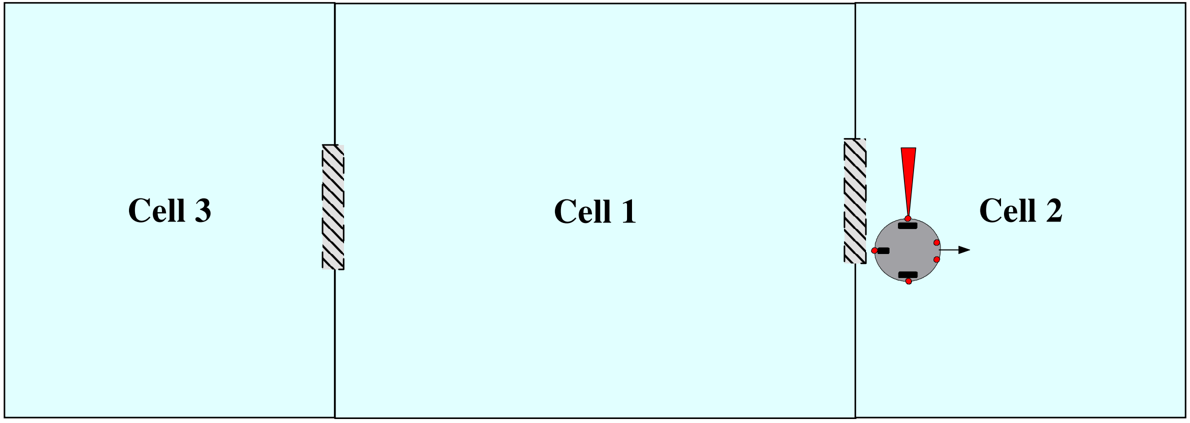


Figure 5.26

- Step 1: In this step we see a robot during its Search behavior inside cell 2. It detects a new cell passage in location $[X2, Y2]$ and decides to pass through it into cell 1.
- Step 2: In this step we see the robot during its Scan behavior in cell 1.
- Step 3: In this step we see the robot during the Search behavior looking for a new cell passage. If there were no odometer errors the robot would have detected that cell passage 1 is not a new one, skip it and continue to cell passage 2. But since odometer errors have been accumulated the location $[X1, Y1]$ (which is the location the robot would have seen when detecting cell passage 1 if there were no odometer errors) now seems to the robot as $[(X1 + error_X), (Y1 + error_Y)]$ and therefore when the robot uses the formula for deciding if cell passage 1 is a new one it will perform the following calculation:

$$Di = \sqrt{((X1 + error_X) - X2)^2 + ((Y1 + error_Y) - Y2)^2}$$

And if $error_X$ and $error_Y$ are too big then the cell passage will be considered as a new one.

- Step 4: In this step we see that because of the accumulated odometer errors the robot mistakenly interpreted cell passage 1 as a new cell passage and passed through it instead of skipping it. This will damage the algorithm performance since the robot will now spend valuable work time in cell 2 which has already been covered instead of operating in a new cell.

The following figure (Figure 5.27) shows a simulator run with and without odometer errors.

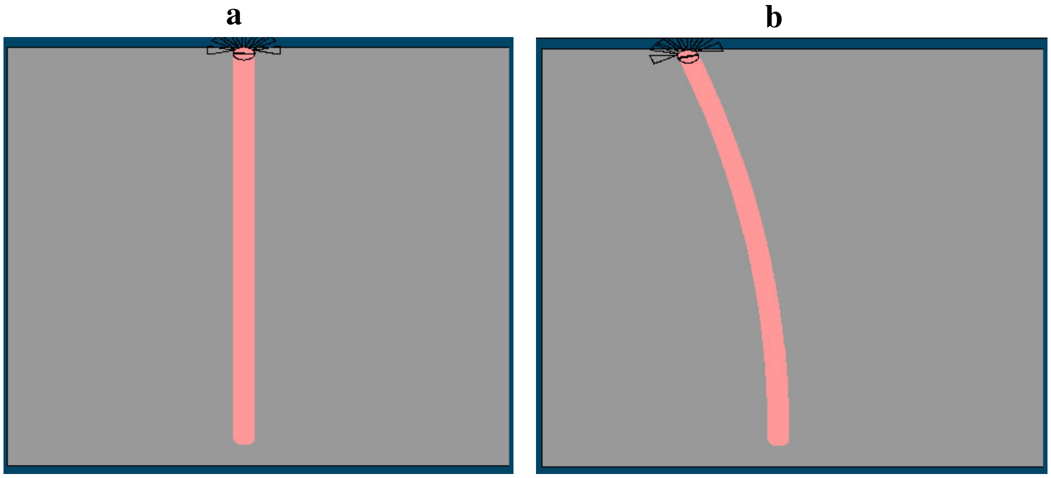


Figure 5.27

- Figure 5.27a: In this option we see the robot performing a straight leg when no odometer errors are inserted by the simulator.

- Figure5.27b: In this option we see the robot performing a straight leg when odometer errors are inserted by the simulator.

In order to test the influence of odometer errors on our algorithm performance we used the simulator to insert random odometer errors. This cause the robot's heading direction to deviate between 0.03°- 0.16° every 1 second. We repeated the tests performed on the office floor in order to compare the results.

The following figure (Figure 5.28) shows the area coverage results for the parallel algorithm.

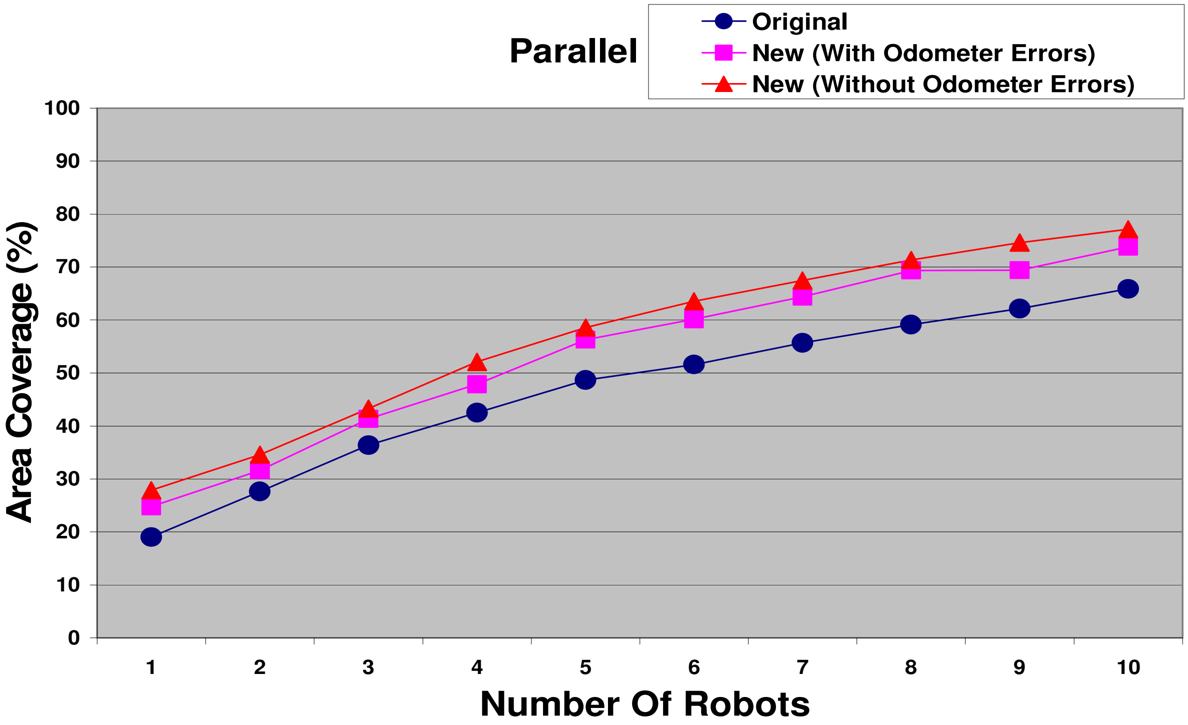


Figure 5.28

A one-tailed t-test performed on the area coverage results for the parallel algorithm shows that there is a statistical significant indication that odometer errors decrease the area coverage in our algorithm (the result is 2.11E-06).

Another one-tailed t-Test performed on the area coverage results for the parallel algorithm shows there is a statistical significant indication that although the odometer errors damage our algorithm area coverage, our algorithm still maintains better area coverage results than the original algorithms (the result is 5.85E-07).

The following figure (Figure 5.29) shows the area coverage results for the random algorithm.

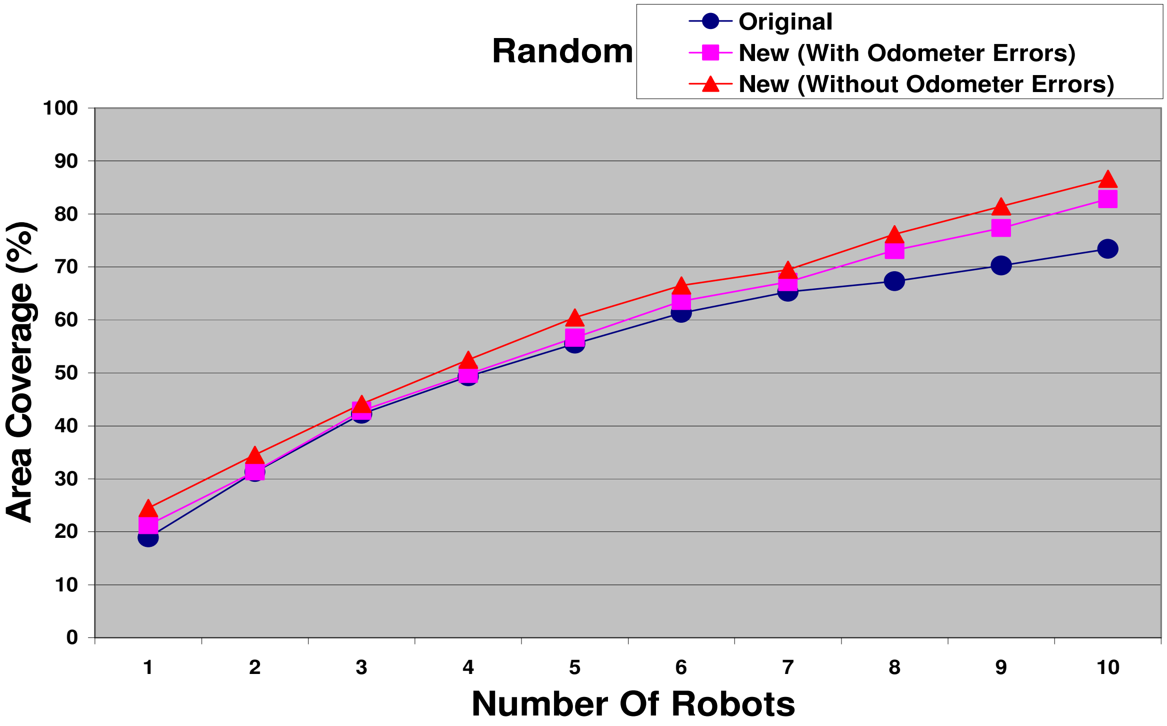


Figure 5.29

A one-tailed t-Test performed on the area coverage results for the random algorithm shows that there is a statistical significant indication that odometer errors damage the area coverage in our algorithm (the result is 4.68E-07).

Another one-tailed t-Test performed on the area coverage results for the random algorithm shows there is a statistical significant indication that although the odometer errors damage our algorithm area coverage, our algorithm still maintains better area coverage results than the original algorithms (the result is 0.006).

The following figure (Figure 5.30) shows the area coverage results for the spiral algorithm.

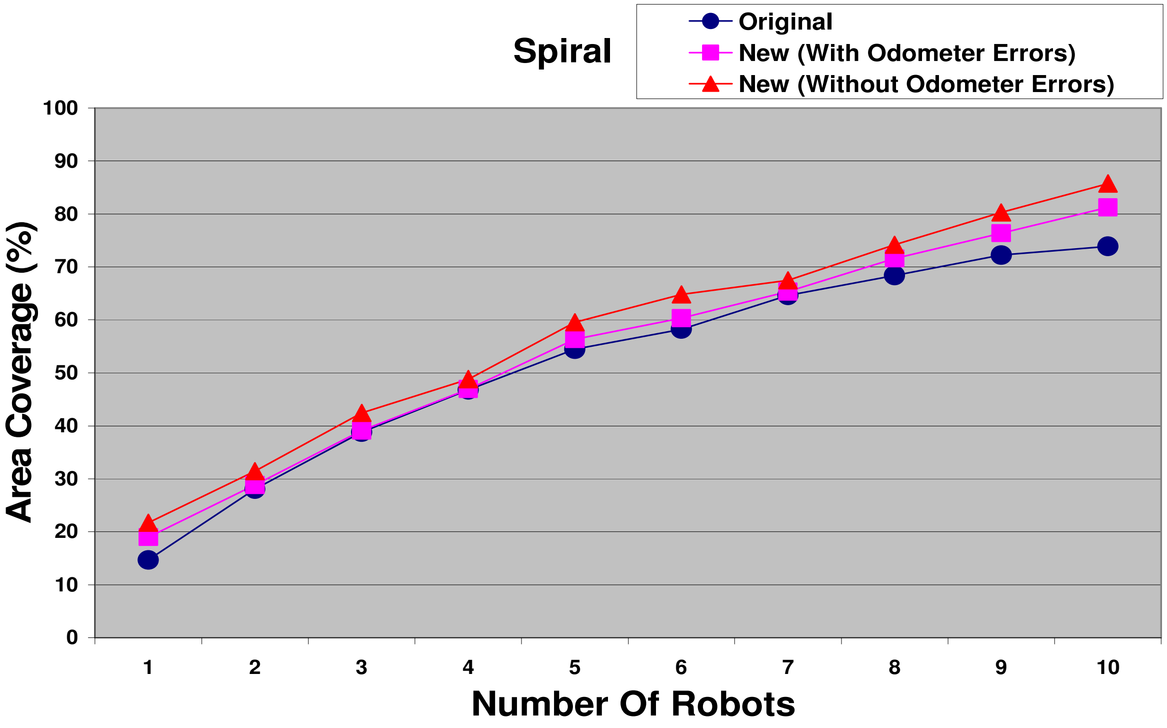


Figure 5.30

A one-tailed t-test performed on the area coverage results for the spiral algorithm shows that there is a statistical significant indication that odometer errors damage the area coverage in our algorithm (the result is 1.04E-06).

Another one-tailed t-Test performed on the area coverage results for the spiral algorithm shows there is a statistical significant indication that although the odometer errors damage our algorithm area coverage, our algorithm still maintains better area coverage results than the original algorithms (the result is 0.003).

The following figure (Figure 5.31) shows the standard deviation results for all algorithms.

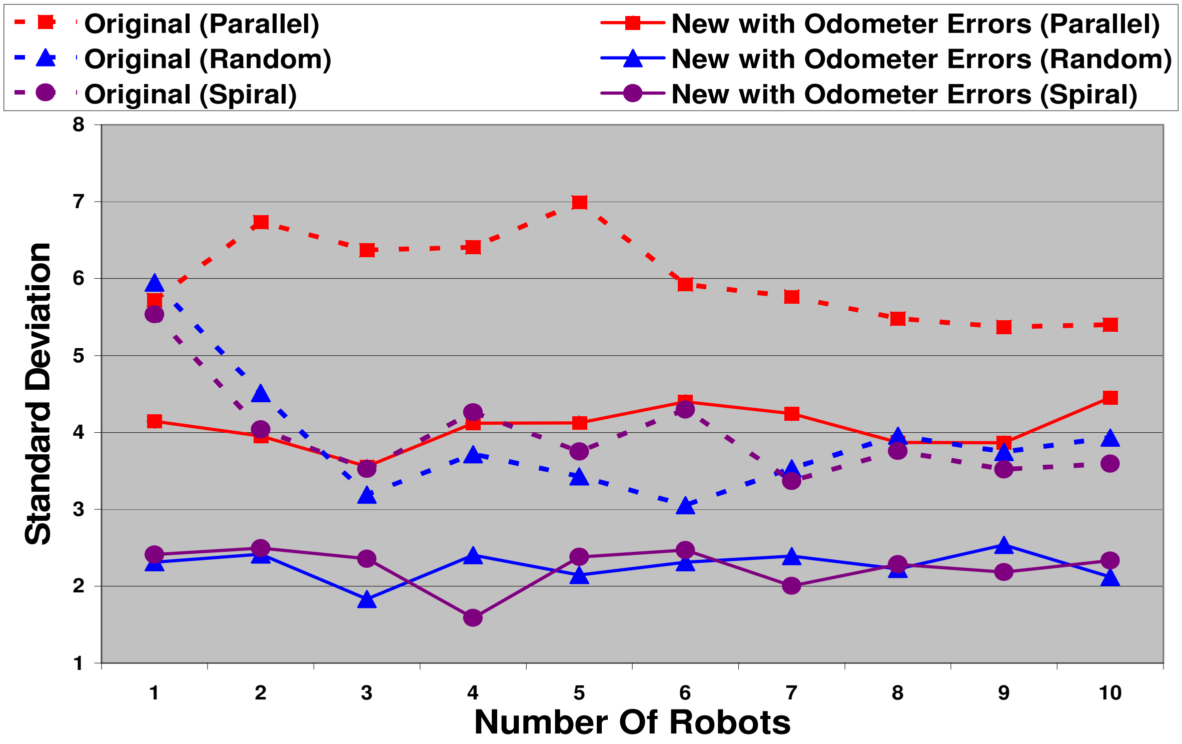


Figure 5.31

A one-tailed t-test performed on the standard deviation results for all algorithms shows there is a statistical significant indication that despite the odometer errors our algorithm still maintains better standard deviation results than the original parallel (the result is 4.26E-06), random (the result is 6.05E-05) and spiral (the result is 8.81E-06) algorithms.

As we can see the odometer errors do damage the performance of our algorithm, but even after entering significant random odometer errors our algorithm produced better results than the original ones. Anyone that will use our algorithm will have to perform real tests on the robotic platform which is used in order to decide if the odometry errors damage in the results is tolerable. Also there is always the option of using other type of technology which is not as sensitive as the odometry system. For example if the cell passages could be marked with some kind of bar-code stickers, then once a bar-code reader will be assembled on the robot we could distinguish between different cell passages without being influenced by the odometry inaccuracy.

5.8 Transmitting cell passages locations

In the previous section we discussed the option of using a more accurate system like the bar-code idea in order to mark the cell passages locations. If such a system is implemented there could be other parts in the algorithm we could improve. For example if the robotic platform which use our algorithm is equipped with RF (Radio frequency) communication or any other type which enables robots located in different cells to communicate, then we could improve our algorithm by creating a communication protocol where each robot transmits the bar-code numbers of the cells it has been in, causing other robots to look for new cells that hadn't been covered instead of covering the same cells over and over again, and therefore improving the area coverage performance. We decided to run a simulation to check how our algorithm performance is affected by such a change. We tested all algorithms on the office floor and added the following change to the algorithm: Every time a robot goes through a new cell passage in order to cover the cell it will transmit this cell passage bar-code number to all its other fellow robots in order to enounce this cell had been taken care off.

The following figure (Figure 5.32) shows the area coverage results for all algorithms.

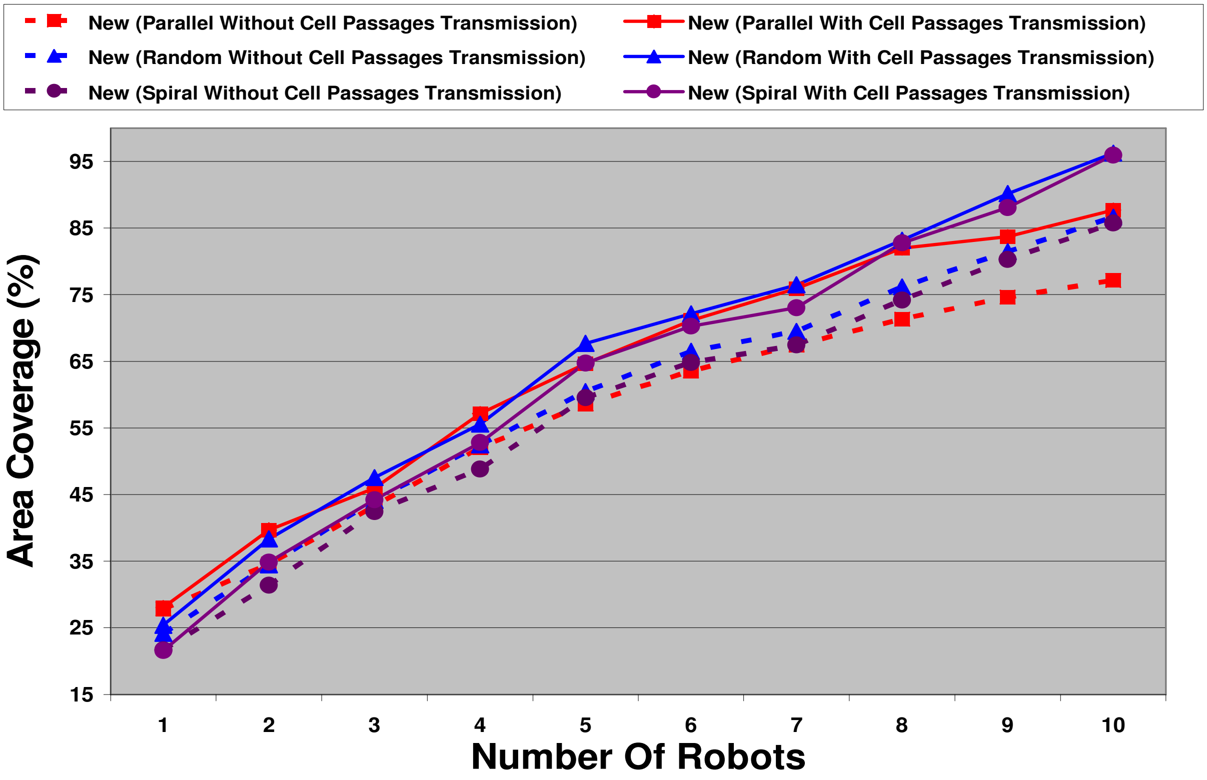


Figure 5.32

A one-tailed t-test performed on the area coverage results for all algorithms shows there is a statistical significant indication that cell passages transmission improves our algorithm area coverage (The result is parallel is 9.03E-05, the result in random is 4.70E-05 and the result in spiral is 0.0002).

As we can see transmitting the cell passages locations improves our algorithm performance.

6 Conclusions and Future Work

The work in this thesis is about a new algorithm which is a black box that can be integrated into many of today's robotic platforms without performing any far-reaching hardware modifications. This will allow converting these robots from a single robot application to a multi robot application solely by changing the robot's software. Our algorithm uses special designed behaviors and information sharing in order to cause the robots to spread out over the environment to maximize coverage, and reduce the variance in repeated applications.

Experiments with different area coverage algorithms, different test areas, different number of robots and real world errors such as odometry errors were performed in order to test our algorithm's performance. Experiments were performed using a simulator that was specially developed.

Future work for continuing this research would be:

- To test the algorithm on real physical environment using real robots.
- To see if the results keeps improving as more and more robots are added.
- Test our algorithm on more area coverage algorithms.
- Test more types of maps: different shapes and sizes of obstacles, round cells, etc...
- Test how different ID transmission range influences the results, and furthermore, test more types of communications technologies.

Bibliography

[1] Macedonia, R.M., and B. Sert. "Low Cost Mobility for Robotic Weapons", AUVS-88, the Fifteenth Annual AUVS Technical Symposium, San Diego CA, 6-8 June 1988.

[2] Flynn, A.M. "Gnat Robots (And How They Will Change Robotics)", Proceedings of the IEEE Micro Robots and Teleoperators Workshop, Hyannis MA, 9-11 November 1987. Also appeared in AI Expert, December 1987, p 34 et seq.

[3] Miller, D.P. "Multiple Behavior-Controlled Micro-Robots for Planetary Surface Missions", Proceedings of the 1990 IEEE International Conference on Systems, Man and Cybernetics, Los Angeles CA, November 1990, pp 289-292.

[4] Bonasso, R.P. "Creature Co-Op: Achieving Robust Remote Operations with a Community of Low-Cost Robots", Fifth Conference on Artifical Intellgence for Space Applications, Huntsville AL, May 1990, pp 257-269.

[5] Flam, F. "Swarms of Mini-Robots Set to Take on Mars Terrain", Science, 18 September 1992, p 1621.

[6] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In AIAA/NASA CIRFFSS, 1994.

[7] J. Nicoud and M. Habib. The pemex autonomous demining robot: Perception and navigation strategies. In Proc. of IEEE/RSJ Int. Conf. on Intelligent Robot Systems, pages 1:419.424, 1995.

[8] H. Choset. Coverage for robotics—a survey of recent results. Annals of Mathematics and Artificial Intelligence, 31:113–126, 2001.

[9] Friendly Robotics®, Ltd. Friendly robotics vacuum cleaner. http://www.friendlyrobotics.com/friendly_vac/.

[10] T. Balch and R.C. Arkin, Communication in reactive multiagent robotic systems, Autonom. Robots 1(1) (1995).

[11] D. MacKenzie and T. Balch, Making a clean sweep: Behavior based vacuuming, in: AAAI Fall Symposium, Instationating Real-World Agents (1996).

[12] I. A. Wagner and A. M. Bruckstein, Cooperative Cleaners – A Study in Ant-Robotics in Communications, Computation, Control, and Signal Processing: A Tribute to Thomas Kailath (Kluwer Academic, Dordrecht, 1997) pp. 289–308.

[13] M. Jager and B. Nebel. Dynamic decentralized area partitioning for cooperating cleaning robots. In Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA), pages 3577--3582, Washington, DC, USA, 2002.

[14] S. V. Spires and S. Y. Goldsmith. Exhaustive geographic search with mobile robots along spacefilling
curves. In Proceedings of the First International Workshop on Collective Robotics, pages
1–12. Springer-Verlag, 1998.

[15] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. Cooperative sweeping by multiple mobile robots.
In Int. Conf. on Robotics and Automation, 1996.

[16] Noa Agmon, Noam Hazon, and Gal A. Kaminka. Constructing Spanning Trees for Efficient Multi-Robot Coverage. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06), 2006.

[17] Noam Hazon and Gal A. Kaminka. Redundancy, Efficiency, and Robustness in Multi-Robot Coverage. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA-05), 2005.

[18] Noam Hazon, Fabrizio Mieli, and Gal A. Kaminka. Towards Robust On-Line Multi-Robot Coverage. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06), 2006.

Appendix A: Glossary

- **Ultrasound**: A procedure in which high-energy sound waves (ultrasound) are bounced off some obstacle and make echoes
- **IR**: Infrared. Invisible electromagnetic radiation used for many purposes, such as night vision and targeting advanced weapons.
- **RF**: Radio frequency. frequency of the radio waves on which a transmission is broadcast.
- **MCU**: Microcontroller Unit