Robotics Systems: Coverage Challenge

Aiyu Liu[1], Boji Wu[2], Ning Bei[2], and Xiyan Zhang[2]

[1]Department of Computer Science, University of Bristol
[2]Department of Engineering Mathematics, University of Bristol

1 Introduction

For CW2 we have chosen to investigate the 'coverage' topic. The coverage challenge involves an unknown map preset with obstacles, after which the Romi has to cover every point of this map, while avoiding these obstacles along the way. Firstly, we implemented a random walking behaviour, which served to be our baseline solution. Subsequently, we implemented the wavefront algorithm. Experiments were then conducted in order to assess the performance of these algorithms in relation to the full task scenario. In this report, we will present our experiments and findings to suggest that wavefront achieves slightly better performance compared to random walk. By doing this comparison, we have found factors that influence the overall coverage rate.

2 Introduction to algorithms

Our two main "walking" algorithms, i.e. methods for covering the map, are the random walk and the wavefront algorithm.

Random Walk

This algorithm makes the Romi walk around the map randomly, while being able to do internal mapping of the environment, avoid obstacles and detect the boundary of the map.

Wavefront

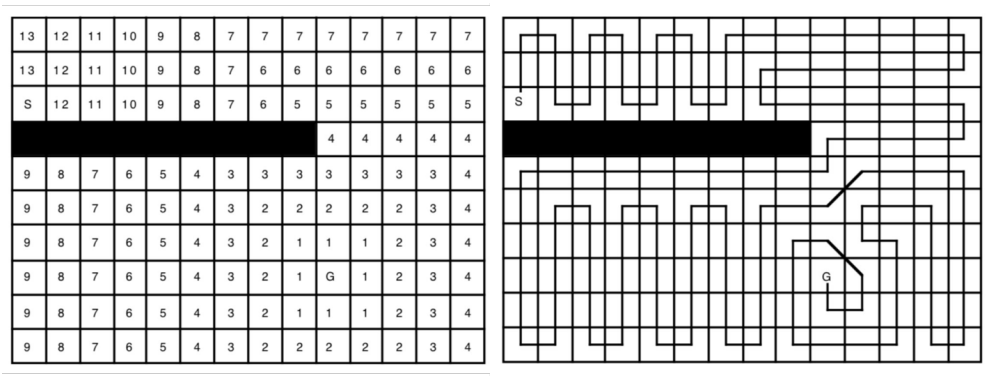


Figure 1: Wavefront in theory

As illustrated, the idea behind wavefront is that the Romi maps the environment with numbers and then follows along the path of which a certain number has laid out. Once said number has been covered, it traverses the path of a lower number. This implies that the Romi can alter its way of covering the map by simply modifying its internal map layout.

As long as the Romi follows its path, going outside of bounds is none of our concern. Furthermore, if the obstacles are mapped correctly with its equipped sensors, it should be avoiding obstacles effortlessly.

3 Implementation and code design

Random Walk

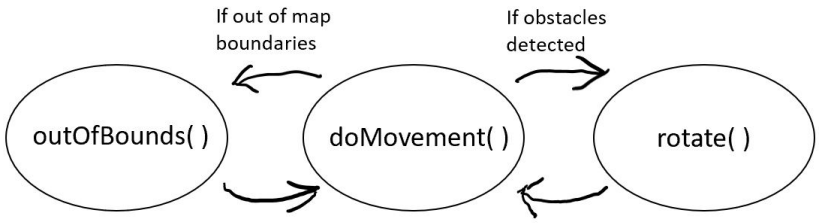


Figure 2: Random walk state machine

This algorithm makes the Romi move forward with a preset speed in addition to a turn bias. The turn bias is based on the *randGaussian()*, which is updated every 500 milliseconds. With 3 sensors, it is able to do an internal mapping of the external environment as well as do obstacle avoidance. Once an obstacle is detected it rotates 1 second, *rotate()*. Using *Kinematics*, it is able to detect whether or not it is currently outside the map. If detected, it shifts to a similar rotation behaviour, this time the Romi moves straight forward for 0.5 seconds after a rotation of 1 second, *outOfBounds()*.

Wavefront

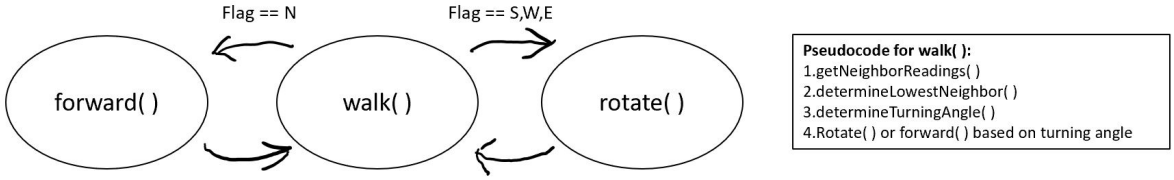


Figure 3: Wavefront state machine

getNeighborReadings() reads the 4 neighboring grid values: N, S, W, E of the Romi, depending on which way the Romi is facing. *determineLowestNeighbor()* goes through each of the 4 readings and finds the lowest one by *char* comparisons. In the case that there is no lowest neighbor, it moves forward. *determineTurningAngle()* determines the target degrees for which the Romi should rotate, based on the lowest neighbor and the current heading.

In our implementation, the Romi begins with walking forwards one step, *forward()*, after which it enters *walk()*. This is not the best approach, since there could be an obstacle right in front of us. Instead it might have been better to just go into walk directly (can be changed in the future easily).

irproximity.h

In the case that the Romi drives straight and there is an obstacle diagonally in front of it, it would not be able to avoid this obstacle if solely relying on one middle sensor. That is why, we chose to have three sensors. This can be seen in the videos in our google drive, referenced later on.

This library is crucial for obstacle avoidance, as the sensors provide the only communication with the external world. So we did experiments to determine in which circumstances they would produce the best readings. Firstly, we did a power regression with the readings(y) as a function of the distance away from an obstacle(x) and found: $y = 2583x^{-0.643}$. This was determined with a Kalmann filter of $\alpha = 0.15$ in order to reduce noise. In the report, we will use α as the parameter for the Kalmann filter.

We have written a script that prints the largest and smallest values every 5 seconds, after which they were reset. Using, the Serial Monitor we recorded these values in the table below:

Dist(cm) \ Alpha	0.4	0.2	0.1	0.02	0.01
5	52/55	53/55	54/55	54/55	55/55
10	81/88	76/82	80/82	86/86	85/86
15	133/145	135/142	137/142	140/143	139/139
20	178/207	185/194	196/203	200/202	200/201
25	242/263	242/257	246/255	254/257	255/257
30	283/324	295/314	296/306	300/308	310/312
35	323/380	346/372	260/75	357/363	353/355

Table 1: The min/max values (mm) at certain alpha and distance values

However, later on we realised that the above outputted values were based on the previous regression result($y = 2583x^{-0.643}$ based on $\alpha = 0.15$). As such the outputs were not true to the tested alpha values (biased towards $\alpha = 0.15$). **Ideally, it would have been best to not make use of the drawn regression when doing this experiment. And instead draw a new regression based on the newly chosen** α. It is worth mentioning that we only did this for the middle sensor, but it would have been worthwhile to do it for the left and right sensors as well, since each sensor is different.

Firstly, our experiment shows that the smaller the alpha-value the more accurate our distances are (the past distances are more higher). However, for smaller alpha values, the delay for rising to a stable value takes longer. We thought $\alpha = 0.01$ was the best one, so we used this with random walk. Later, we found that using $\alpha = 0.15$ achieved a better balance between accuracy of readings and delay to a stable value (used in wavefront). In order to account for the aforementioned delay, the IR calibration method gets rid of the first 1000 readings.

Secondly, our experiment shows that the IR-sensor reaches better accuracy at slightly higher values, which is why our obstacle mapping range is set higher (begin mapping at 150-200 cm), while our obstacle avoidance are inclusive of smaller distances (begin avoiding at 80-150 cm). For wavefront, we begin mapping from 164-200 cm.

mapping.h

The advantage of the wavefront algorithm is that we are able to alter the walking behavior based on the internal map representation. For examining different coverage performances we have produced these 3 different walking maps for coverage of 27x27 each:

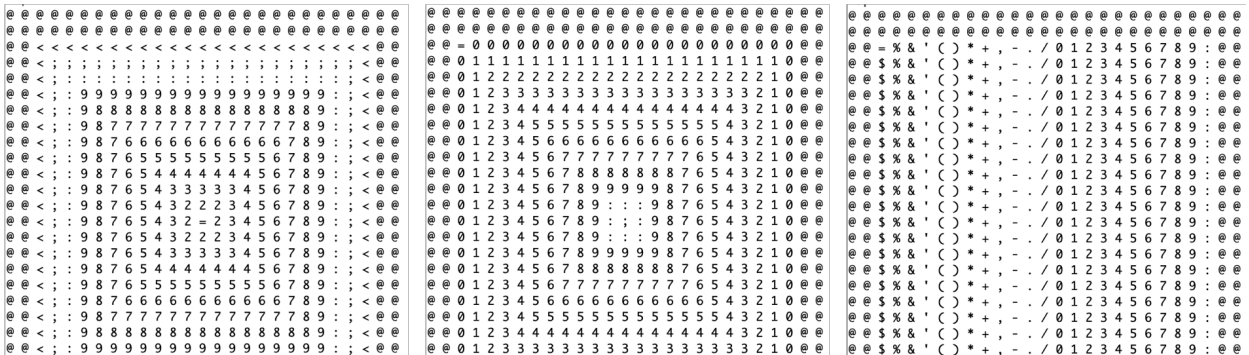


Figure 4: Internal map representations of three walking methods: circular, inverse-circular and ox-turning respectively.

To avoid getting too close to an obstacle, we have created the functionality to add buffers around an obstacle for safer obstacle avoidance. If one obstacle is detected, we have conservatively chosen to add buffers ('?', relatively higher char value) surrounding the obstacle ('O'). This could have been done differently by adding one buffer in front of the obstacle depending on the heading of the Romi and which sensor detects the obstacle. However, we chose the former option as the obstacle are quite big and needs to be cared for conservatively. Furthermore, if a grid has already been covered we do not draw the buffer on top, as this would erase our coverage and also we cannot fully rely on the sensors.

To avoid going outside of the map boundaries we add two layers of buffers on the outside of the map ('@', highest char value). The first layer is to ensure that the Romi can read its 4 surroundings when it is walking on the border. The second layer consititutes the last cell grid of a 25x25 resolution; making the Romi start from 1 grid and end 1 grid inside this resolution. This is so as to make the Romi walk a lesser distance when doing ox-turning and inverse-circular.

kinematics.h

In this library we have mainly modified the behavior for rotation. Herein, we mainly had to circumvent the issue of sudden change from 180 to -180, which resulted in the Romi not being able to rotate from -180 to 90(1) and 180 to -90(2). To circumvent this we created buffers. I.e. for (1) we set a new limit of -185 and change to 175 if theta drops below this limit. For (2) we allow the opposite: $185 - 175$. These buffers are necessary since, they would change the PID controller behaviors, where we are using a forward heading control, speed control and rotation heading control. We aslo found that by scaling our theta by 0.98, we achieved better rotation. Lastly, the kinematics update is dependent on the execution time of the code, so we decided to do independent updates by moving it to a timer of 100 Hz.

4 Experiments

The videos for our walking (only for map with obstacles) can be found through HERE. **Each experiment is limited to 3 minutes.**

Outline of experiments

1. **Baseline: Random walk**
   (a) $speed = 6$, $turnBias = 0 - 6.5$, $rotationSpeed = 10$, $\alpha = 0.01$
2. **Map with no obstacles: Circular walking → Inverse circular, Ox-turning.**
   (a) $speed = 6$, $rotationSpeed = 7$, $\alpha = 0.15$
3. **Map with obstacles: inverse-circular and circular**
   (a) same as experiment 2.

Experiment Metrics

As our topic choice suggests, we need a way for us to measure the performance of how well we have covered the map.

Internal map coverage (IMC)

We have defined our map to be 27x27. However, using 25x25 we define the following from the romis internal map: $coverage_{IMC} = \frac{number\ of\ '='}{25 \cdot 25}$. This is rather unreliable as we know that while the Romi thinks it is behaving normally, in reality, we can see that it drives inaccurately. This metric is thus quantitative but not reliable to the real world coverage. **Furthermore, it would be valuable to experiment with lower resolutions as this would likely improve obstacle avoidance leading to higher overall coverage**. I.e. the cell grids are bigger making the obstacles easier to avoid

Real map coverage (RMC)

To circumvent the above inaccuracy issue, we need to introduce a realworld coverage metric. We could physically draw the map into 25x25 cells of length $1800/25 = 72$ cm each (or use a drawing software) and record the number of covered cells. However, this method is rather time consuming. Instead, we thought of making the Romi draw its path by itself and thereafter manually counting the cells it has covered. This was done by attaching a pen behind the back of the Romi[1]. Friction was not a concern, since the pen drew smoothly. One has to avoid making the Romi tip forward with the pen, as this influences sensor readings. Also, the pen is quite easily stuck on the carpet when the Romi goes outside the map.

However, when calculating the coverage using the resulting RMC image, it takes a lot of effort to count the cells due to the attachment to the back of the Romi. It would have been many times easier if this attachment was able to be made below the center of the Romi. In the end, we were not able to use this metric for quantitative measures and only for accuracy. That is, by comparing the IMC and RMC we can

[1] P.S. we were the first ones to use this metric

clearly check for inaccuracies.

Obstacle hit (OBH)
Hitting an obstacle has significant impact on the coverage amount. The longer the Romi spins its wheels at an obstacle, the greater the coverage amount is affected. As a result, the coverage amount has been artificially increased even though it spins its wheel in one grid. We denote this spinning time by $t_{obstacle-hit}$ in seconds. To account for this we have decided to calculate the final coverage in the following way:

$$coverage\_rate = \frac{(number\ of\ ' = ') - t_{obstacle-hit}}{25 \cdot 25}$$

.

Ideally, we would want to use a RMC metric, but we had to fall back on the the IMC metric modified with time stuck to an obstacle.

Outside of bounds (OOB)
Going outside of the map boundary is undesirable, since we do not want to cover an area which was not originally intended. So if this behavior is detected to be high, it implies poor coverage. So we set a timer for how long the Romi spends outside of the map boundary.

Experimental methodology
Before we did anything we decided to experiment with the sensors in order to get the most accurate readings (see section irproximity.h above).

(1) We decided to further develop the provided random walking algorithm, seeing that it would not be fair to do comparisons with the provided baseline solution. Avoiding obstacles is also one of the objectives of the original coverage challenge, so this experiment was done with obstacles. This serves to be our baseline solution which we will be doing our comparisons with.

(2) We specifically decided on the circular map, since we thought it was mandatory to start from the center. As we solely wanted to test our implementation, we started with very low environmental complexity. I.e *no obstacles*. We found that the rotation would progressively get more inaccurate. In line with the paper [1]: *"The high number of turns and path segments will inevitably cause errors to be introduced in the estimation of the robot's correct position. Possibly for these reasons an implementation of complete coverage paths on an actual mobile robot has not been reported."*, we thought that we could improve the accuracy by altering the rotation and driving straight style. Hence the reason why we chose to do the ox-turning and inverse-circular mapping; all three methods have the same turn-amount of 44 but different path segments.

In the beginning, we found that ox-turning had some issues, so we disregarded this method for experiment 3. Towards the ending of the project, we found that, for some mysterious reason, it was then able to do its intended behavior. **Due to time limitations, we decided to only do exp. 2 with ox-turning and not exp. 3.**

To further determine where the rotation error was coming from, we collected the theta values by walking with the Romi using circular method. Then we took the very last reading before it shifts to another behavior: forward or rotate. That is, we take the last reading of the first 25 behaviors, first 14 forward behaviors and last readings of the last 14 behaviors. We then calculate the sum of the differences between the target and output theta values, making this the error value.

Method / Data	Left Rot.($speed = 6$)	Left Rot.($speed = 12$)	Right Rot.($speed = 6$)
First 25 Behaviors Error	4.99°	4.96°	5.60°
First 14 Forward Behaviors Error	1.63°	1.19°	0.70°
Last 14 Forward Behaviors Error	20.85°	2.18°	0.56°

Table 2: Theta value errors for behaviors(rotate or forward)

Although, the empirical evidence is not strong (low experiment repetition), we can say that the kinematics is not reliable. This is evident from the first column, where we had the highest accuracy but recorded the highest error. **Furthermore, we found that higher speeds induces greater inaccuracy, although higher coverage**. We chose to go with lower speed, since accuracy is extremely important for avoiding obstacles. Though, it would be interesting to experiment with a middleground.

We did the first attempt of circular method to which the RMC was very accurate. However, the

remaining ones performed significantly worse. As these were done on a separate day with days of Romi use in between, we suggest that this is very likely to be a cause of **insufficient battery levels**.

From the other two walking methods, we found that driving in a straight line is another cause to the inaccuracies. That is, **if the turning is off by a certain degree, then driving forwards in a straight line further enhances the previous error.**

(3) Lastly, we increase the environment complexity by adding obstacles. Now, we can compare results with the baseline solution as obstacles exist for both. Here we found that an $\alpha = 0.15$ and restricting obstacle/buffer remapping provided much better results for obstacle avoidance. We are able to do some obstacle avoidance but it is far from perfect. This is highly likely to be due to the IR-sensor errors as our internal map representations does not provide 1:1 equality to the real world obstacles. Also, as small errors in kinematics accumulate, we are also prone to hit obstacles.

5 Data

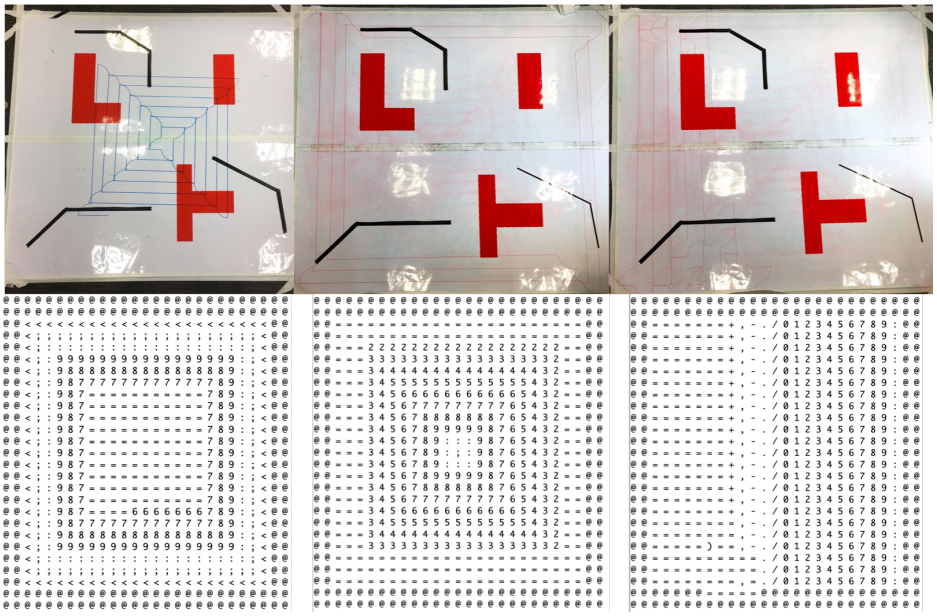


Figure 5: Comparisons between RMC and IMC

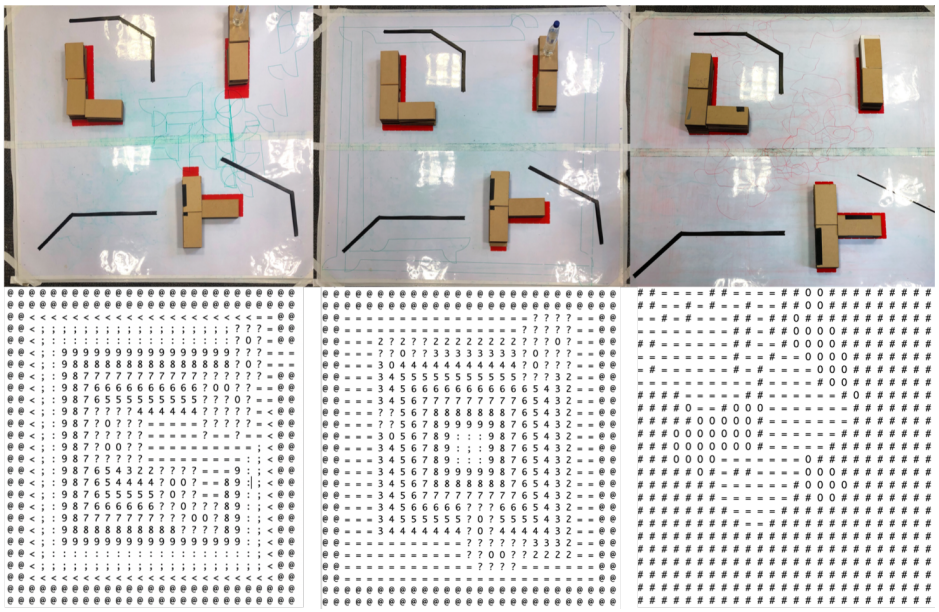


Figure 6: Comparisons of RMC and IMC with obstacles

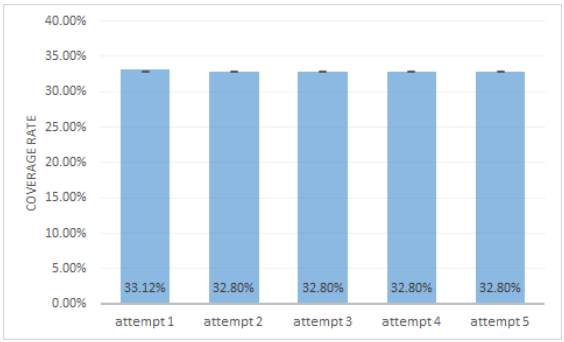
Examples of the above walking behaviors (with obstacles) can be found HERE.

As mentioned before, we did not draw the RMC into real-world grids, thereafter counting the number of grids, mainly due to the pen attachment to the back of the Romi. It would be more manageable to have the pen attached under the Romi center and doing it by video would also be too time consuming. Based on these comparisons we can see that although, the IMC is not hundred percent accurate, it is still able to show the coverage amount to a good extent.

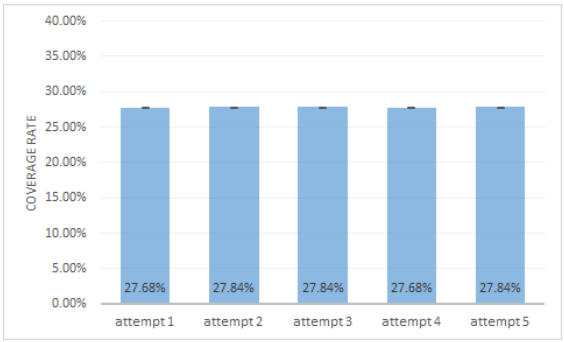
5.1 Exploration Data

OBH / attempt #	Inverse Circular	Circular	random walk	random walk
				OOB
1	0	2s	0	12s
2	0	0	0	29s
3	0	0	0	8s
4	2.2s	2.3s	0	23s
5	1.5s	0	1s	22s

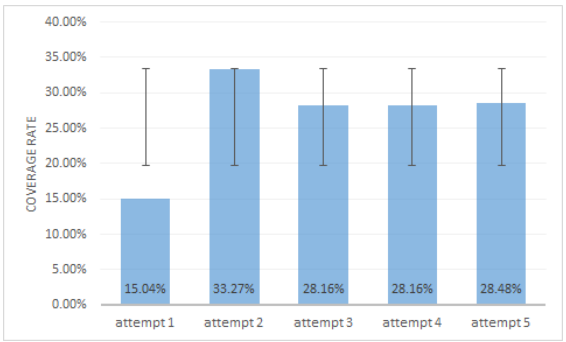
Table 3: Map with obstacle data. Time being stuck to an obstacle for Inverse circular, circular and random walk. Random walk was the only method going outside of map bounds, hence why it has its own column.



(a) Inverse circular coverage rates

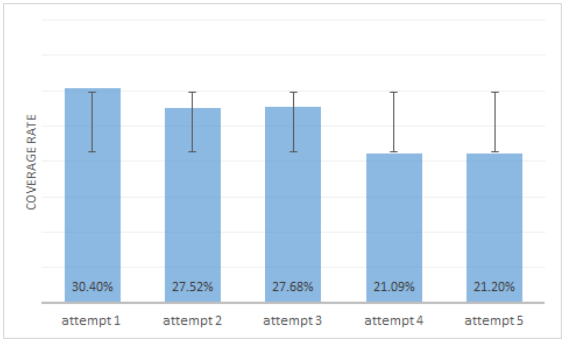


(b) Circular coverage rates

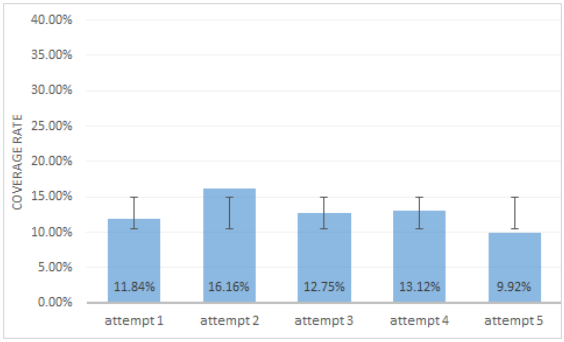


(c) Ox-turning coverage rates

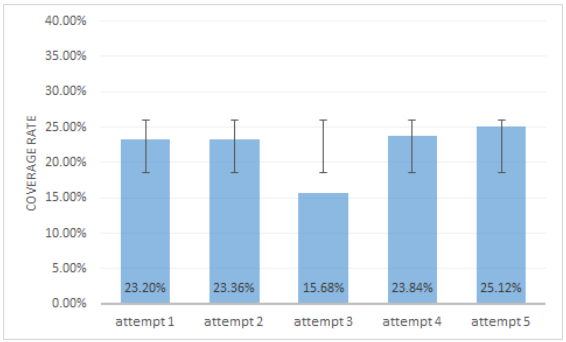
Figure 7: Coverage rates WITHOUT OBSTACLES for inverse circular, circular and ox-turning method



(a) Inverser coverage rates



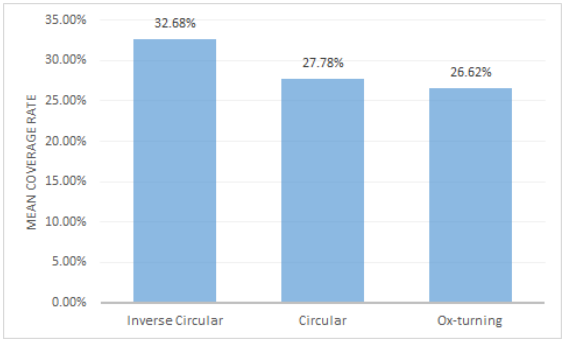
(b) Circular coverage rates



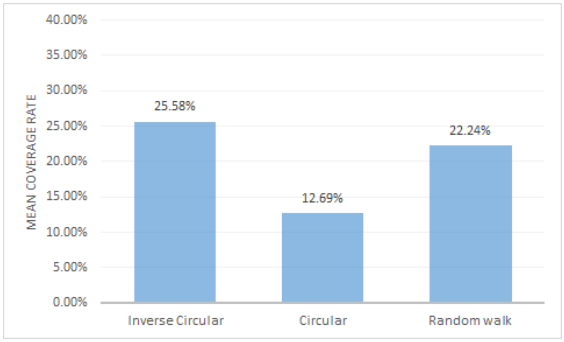
(c) Random walk coverage rates

Figure 8: Coverage rate WITH OBSTACLES for inverse circular, circular and random walk

5.2 Data-processing



(a) Mean coverage rates WITHOUT OBSTACLES



(b) Mean coverage rates WITH OBSTACLES

Figure 9: Mean coverages

Below is our data processing in table format where the relative coverage ratio is calculated as $\frac{coverage_{wavefront}}{coverage_{randomWalk}}$.

Method / Data	Inverse Circular	Circular	ox-turning	Random Walk
AVG Cov. w/o Obs.	32.86%	27.78%	26.62%	–
Standard Deviation of Cov. w/o Obs.	0.143	0.0876	6.83	–
AVG Cov. w/ obs.	25.58%	12.69%	–	22.24%
Standard Deviation of Cov. w/ Obs.	4.21	2.27	–	3.74
AVG OBH time(s)	0.74	0.86	–	0.2
AVG OOB time (s)	0	0	0	18.8
Relative Cov. ratio	1.15	0.57	-	-

Table 4: average data for each exploration data

6 Discussion

Coverage with no obstacles

Starting with ox-turning, the first attempt was done when the Romi behaved abnormally, whereas the remaining four was normal. As a result, the standard deviation is highest of 6.83. Based on the remaining four, we can see that the confidence is quite high. Confidence is also high for inverse-circular and circular. Without obstacles we are thus able to achieve quite reliable performance. The inverse-circular method has the highest mean of 32.86%, which makes sense due to the fact that **it spends less time rotating and more time driving straight, which equates to a higher coverage percentage**. Following this principle, the circular method should in theory be the lowest but it achieves a higher score than ox-turning. This is likely due to the first outlier attempt in ox-turning.

Coverage with obstacles

Before we begin this discussion, it is worth mentioning that in random walk, we did not set the restriction for remapping obstacles/buffers. That is, the mapping is allowed to map covered grids as obstacles or buffers. This was restricted in the other walking methods, and should have been included for random walk as well.

For the real-world obstacle test, we can see that the best performance is achieved with the inverse-circular method. It covers an area that is 1.15 larger than random walk. This is likely due to its natural walking behavior, as it goes inwards and walks completely unobstructed for the majority of its time. That is, there is less obstacle avoidance because the map is biased towards this method. In contrast, the circular walking method is at a disadvantage from the beginning, as it starts from the center and encounters obstacles early on. Thus it covers only about half, 0.57, compared to random walk. We can presumably say that less obstacle avoidance leads to higher coverage.

Also, as expected can see that the standard deviation is significantly higher compared to the experiments without obstacles. This makes sense since the behaviors change once obstacles are introduced. Here, the circular walking is the most affected, dropping from 27.78% without obstacles to 12.69% with obstacles (lowest standard deviation).

One of the advantages of random walk is that its speed can be increased without having too much effect on its behavior. Going faster can likely improve its average coverage rate, although this cannot be determined definitively due to its time spent outside the map and its randomness.

In both inverse circular and circular we have about the same OBH of $0.74s$ and $0.86s$ respectively. The Romi cannot recover once an obstacle is hit, resulting in it driving in circles. This is quite poor performance considering the fact that random walk has half the obstacle hit rate and can even be improved by changing from $\alpha = 0.01$ to $\alpha = 0.15$; in addition to also being able to recover from hitting obstacles.

In terms of going outside of the map boundaries our methods does exceptionally well with not a single OOB being recorded. The opposite is true for random walk with 18.8s spent outside the map. Although, this can be improved by putting more effort into implementation.

We also need to ensure to keep battery levels consistent throughout all experiments as this affects all the interconnected components of the Romi. Due to finance issues, this was not ensured at all times. Consistency in placing the obstacles in all experiments also have a noteworthy effect for testing obstacle avoidance.

7 Conclusion

Below are the pros('+') and cons('-') of the random walk and wavefront:

Random Walk

+ Effective at avoiding obstacles and can recover from OBH.
+ 22.24% coverage amount for a simple solution. It also has higher room for improvement. I.e. not rotating with a delay, and map boundary behavior.
− Unpredictable and has to revisit grids and goes outside of bounds.

Wavefront

+ Driving straight leads to more coverage but higher chance of walking inaccurately. Higher speeds leads to more coverage but greater inaccuracy of RMC.

+ Systematic, logical and reliable performance with no obstacles.
+ Spends all its time covering the map. I.e. does not go outside of bounds.
+ Can be programmed strategically to avoid obstacles if locations are known beforehand.
− Poor obstacle avoidance and cannot recover from obstacle hit due to heavy reliance on the internal map as well as the sensors not being reliable. This implies poor coverage.
− High reliance on battery levels is needed in order to do be accurate in its coverage.

We have found that the wavefront algorithm is very effective and reliable for the coverage challenge without obstacles, achieving a coverage percentage of 32.86% with the inverse circular method.

For a map with obstacles, we are able to beat our implementation of random walk by 15% again using the inverse circular method of 25.58% coverage rate. However, we are not completely able to avoid obstacles, mainly due to IR sensor inaccuracies.

Random walk can be improved with higher speeds and better OOB behavior, while wavefront can be improved with better IR-sensoring or a different obstacle avoidance behavior. So it is not clear which algorithm is best for the original coverage challenge with obstacles. Also our metric for measuring the coverage rate is not completely true to the real world coverage. Furthermore, the number of repetitions for each experiment is quite low, and doing them 10 times would provide higher confidence to our findings.

Further work

It would have served an interesting comparison, if we also did random walk WITHOUT obstacles and ox-turning WITH obstacles, in order to come to further conclusions.

References

[1] Alexander Zelinsky, Ray A Jarvis, JC Byrne, and Shin'ichi Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of international conference on advanced robotics*, volume 13, pages 533–538, 1993.