67301 - MULTI ROBOT SYSTEMS - Spring 2021

Final project presentation

Tamer Ghattas

The Hebrew University of Jerusalem School of Computer Science and Engineering

Single-agent dirt collection

Single-agent dirt collection

- **Problem description:** collect as many as possible dirt pieces from an unknown environment without having the option of sensing the dirt.
- That can be translated to: we need to cover diverse areas in the room in the shortest period of time.
- The problem is hard because with the time limitation we often inherit an upper bound for the area coverage.
- An exhaustive approach to the solution would be moving sequentially along every accessible point known from the map in hand completing a perfect coverage.

Single-agent dirt collection

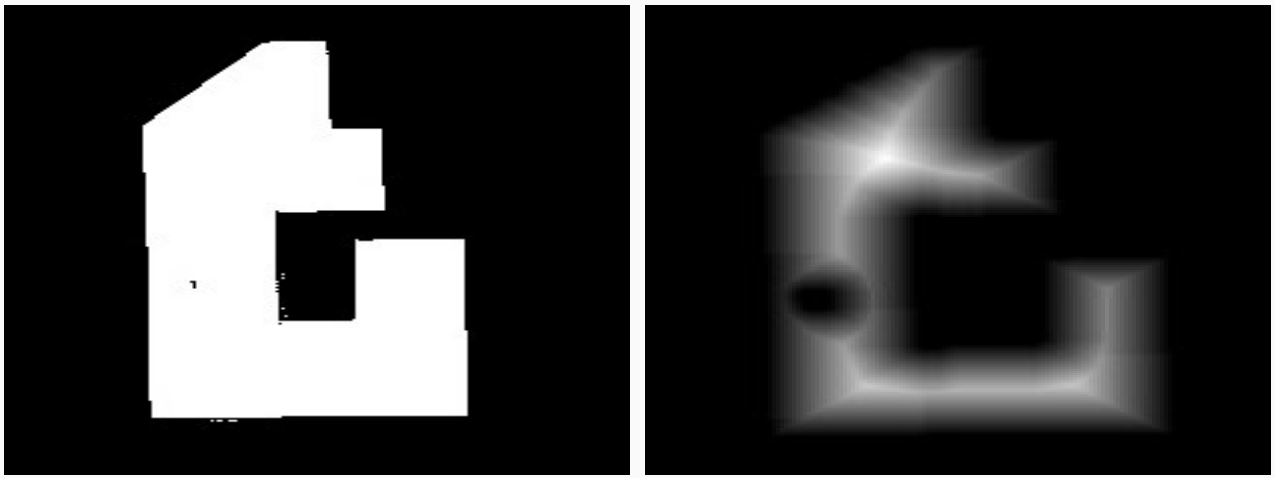
We model the problem as a graph $G = (V, E)$ where the nodes V represent a pixel in the map and E an edge existing between every adjacent pair of pixels (nodes).

Our suggested solution takes the graph as input and derive goals for the robot aiming for diverse area coverage.

Single-agent dirt collection

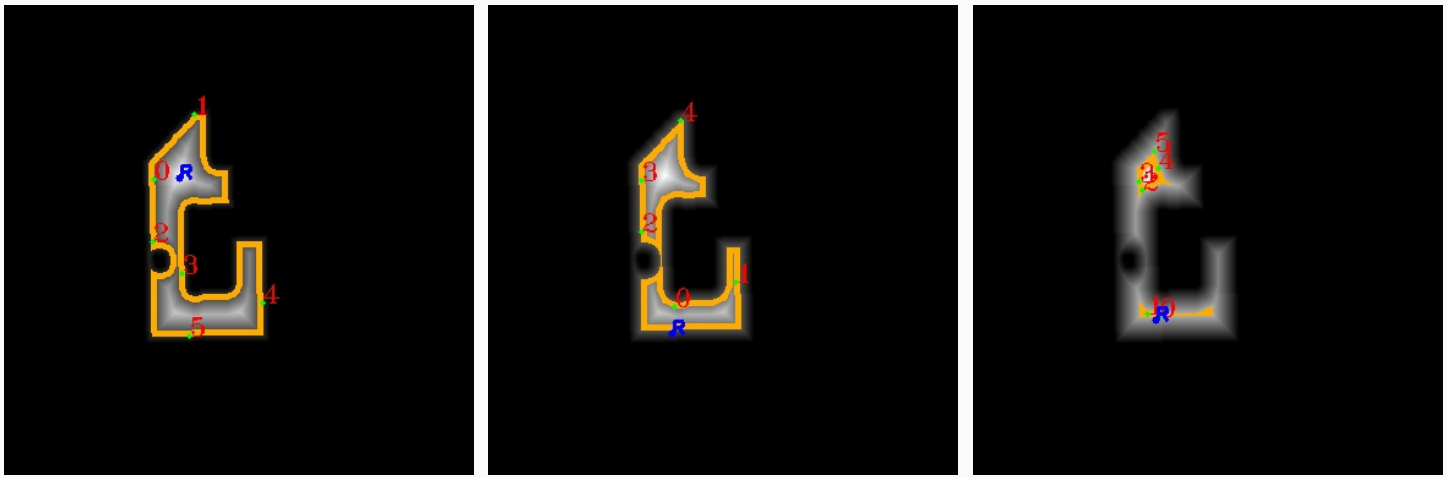
We implemented our approach using image processing manipulations on the given map:

- First we generate binary image from the map structure.
- Then we perform Euclidean Distance Transform on the image.
- Now, we have range of gray levels that are distributed among pixels as a function of their distance of the nearest black pixel, i.e the nearest wall.



Single-agent dirt collection

- By quantizing the EDT image according to grey levels into a selected number of partitions, we get a complete partition of the room.
- Then, we sample from each part, a number of points which are set as goals the robot travel to using move-base node.
- Note that by increasing the resolution of the sampling we can guarantee more area coverage.
- In the following images, we can see the different parts and the sampled goals it yields (in orange):



Single-agent dirt collection

- This approach was the second, first I thought about using Zig-Zag pattern with a randomly sampled angle on each turn but that is prone to missing parts of certain room structures.
- The presented approach on average, covers all parts of the room after enough time.
- A good approach might be to partition the map into compact clusters and run Zig-Zag like pattern on each.

Single-agent inspection

Single-agent inspection

- **Problem description:** explore a room and count the number of statically positioned spheres subject to a time limit.
- That can be translated to: cover all the accessible areas using the agent sensors range and register the spheres positions within the given time frame.
- The problem is hard because, first, exploring is a hard problem and second, detecting sphere using a low quality 2d bird view of the sensors reading is challenging.
- An exhaustive approach to the solution would be using BFS/DFS algorithm to map the room including the existing static objects like the spheres.

Single-agent inspection

We model the problem as a walk-by-wall using PD controller to keep a distance from the wall and gradually increase this reserved distance after each full circle and registering the spheres positioned along the way.

PD formulation:

$$\omega(t) = K_P e(t) + K_P \frac{de}{dt} \quad s.t$$

$\omega(t)$ - PD output

K_P - proportional gain

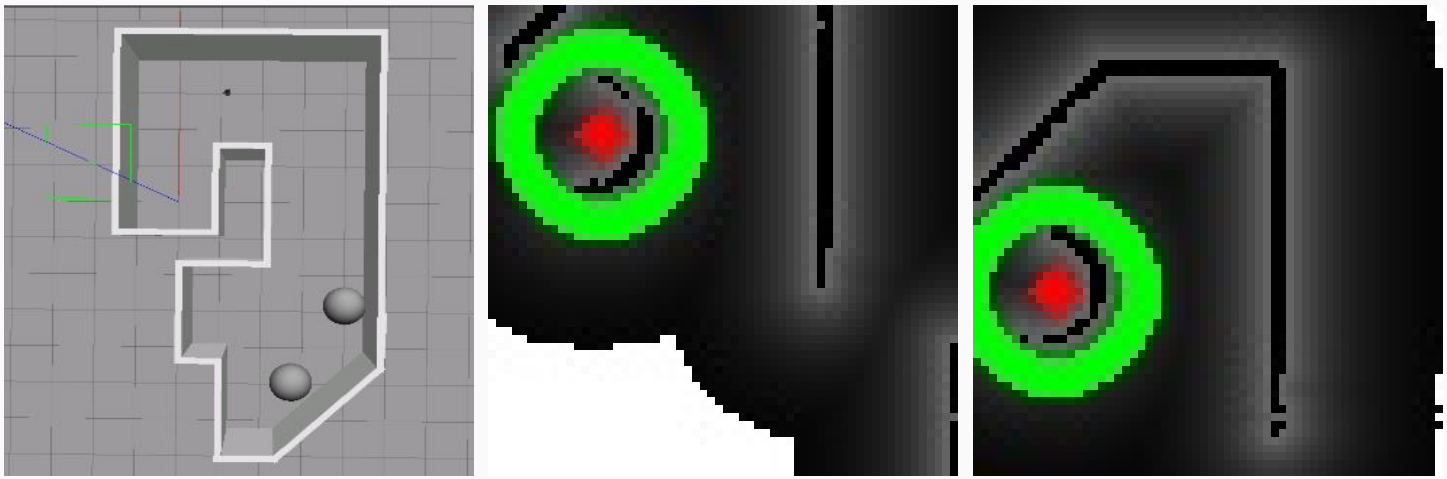
de - change in error value

dt - change in time

Single-agent inspection

We implemented our approach using PD output for movement and the local cost map for spheres count. For every relevant sensor reading:

- Adjust linear velocity according to distance from front wall.
- Adjust angular velocity according PD output.
- Process local cost map image and detect circles using Hough transform.
- For each detected circle center, register centers in a set to prevent counting twice.



Single-agent inspection

- I thought about generating a global map using local cost map registration but that isn't needed for the mere task of counting the spheres.
- The presented approach, did manage to fulfill the task of counting spheres with a miss count of up to 1.
- An improvement to the solution is to break dependency from the wall kept distance.

Multi-agent dirt collection

Multi-agent dirt collection

- **Problem description:** compete against another agent in collect as many as possible dirt pieces positioned in known locations to both agents, with the option of listening to to rival plans.
- That can be translated to: an agent needs to maximize a utility function of the collected dirt pieces number in the existing competitive setting.
- Although one agent can listen to the next step of the other, the problem is challenging because the rival's whole strategy is unknown. Therefore, to gain advantage, the agent strategy has to assume rival level of strength and rationality.
- A safe assumption might be that the rival is rational, and will go in the near optimal path to collect the dirt pieces.

Multi-agent dirt collection

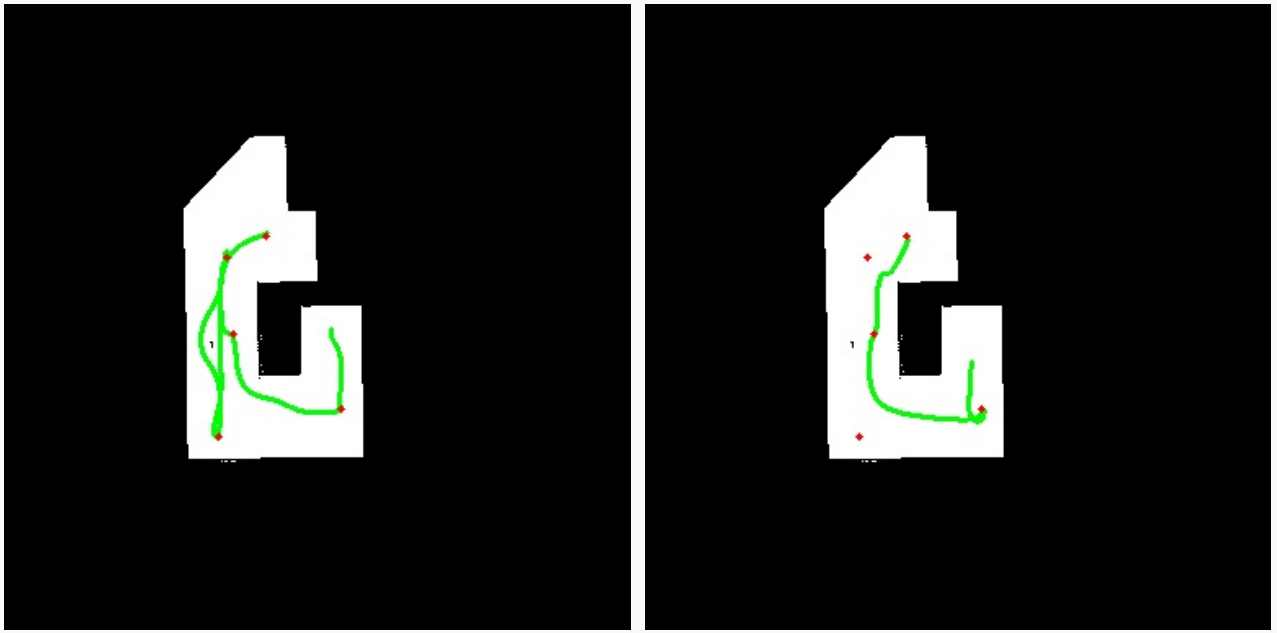
We model the problem as two decision Trees, one for each agent, $T_{Agent_i} = (V_{Agent_i}, E_{Agent_i})$ where:

- Every node is a goal, a physical location of a dirt piece.
- Each parent node in the tree has all the remaining dirt pieces (goals) as children.
- A leaf represents collecting all the published dirt pieces.
- Each edge $E_{Agent_i} = (v_1, v_2)$ cost is the Move-Base path length between v_1 and v_2.
- The utility of an agent is defined for path $p \in E^{\#dirts}$ as
-
$$u(p) = \sum_{e \in p} cost(e)$$
- A path p^* is optimal if
$$u(p^*) = min_{\forall p} u(p)$$

Multi-agent dirt collection

We implemented the approach by:

- Sorting the dirt pieces locations by the length of the Move-Base planned path to goal.
- Listen to the rival goal planing.
- Move to the goals according to the sorted sequence.
- If the agent heading to an impossible to arrive-before-rival goal, skip and go for the next in line.



Multi-agent dirt collection

In some runs, a more naive approach as simply sorting dirt pieces according to euclidean distance from initial location and going sequentially, won the more sophisticated agent. That is probably due to the added overhead of computing the goal as it involves the planner which might be slow.

Multi-agent inspection

Multi-agent inspection

- **Problem description:** explore a room and count the number of statically positioned spheres using two agents subject to a time limit.
- That can be translated to: cover all the accessible areas using the agent sensors range and register the spheres positions within the given time frame.
- The problem is hard because, like with the single agent, exploring is a hard problem and detecting sphere using a low quality 2d bird view of the sensors reading is challenging but what complicates it even more is managing the two agents.
- An exhaustive approach to the solution would be using both agents to perform BFS/DFS algorithm to map the room including the existing static objects like the spheres and cross validate correctness.

Multi-agent inspection

Same as with single agent: We model the problem as a walk-by-wall using PD controller to keep a distance from the wall and gradually increase this reserved distance after each full circle and registering the spheres positioned along the way.

PD formulation:

$$\omega(t) = K_P e(t) + K_P \frac{de}{dt} \quad s.t$$

$\omega(t)$ - PD output

K_P - proportional gain

de - change in error value

dt - change in time

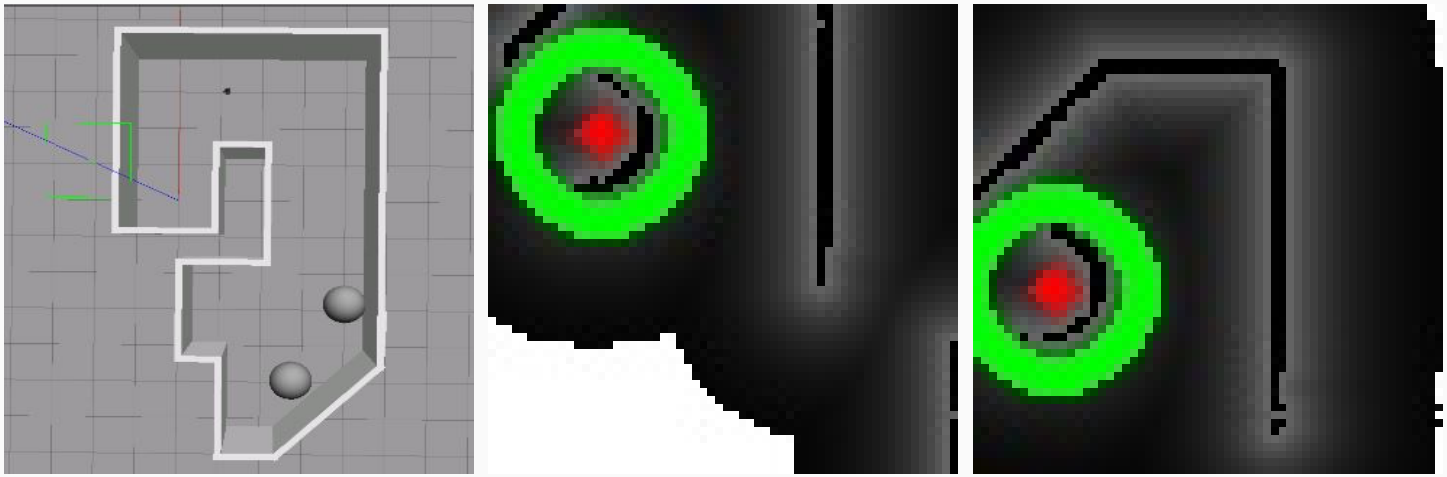
Multi-agent inspection

We implemented our approach duplicating the logic from single agent i.e using PD output for movement and the local cost map for spheres count. However, we make the agents go in opposite directions by rotating one of them before movement and rely on the kept distance from the other wall (left vs right), this ensures concurrent exploring by reducing the overlap.

Multi-agent inspection

For every relevant sensor reading in each agent:

- Adjust linear velocity according to distance from front wall.
- Adjust angular velocity according PD output.
- Process local cost map image and detect circles using Hough transform.
- For each detected circle center, register centers in a set to prevent counting twice.



Conclusion

Conclusion

- Simplicity might come as handy when designing autonomous agents. As having fast reactive controller (e.g. PID) are critical in some tasks.
- When designing solutions in robotics, generalizing solutions is not trivial. It is very task and environment dependant.
- Scaling up solutions to distributed agents might add to the problem if they have limited communications.
- Robots are fun.