236609 - MULTI ROBOT SYSTEMS

Assignment 3 Presentation Template

Adi Amuzig and Moshe Shienman

The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Multi-Agent Dirt Collection

Problem Description

- Two autonomous agents compete in a cleaning task given a fully mapped environment
- **Goal:** collect more pieces of dirt than your adversary agent in two minutes

Problem Description (continued)

- This problem is relevant to real world applications such as: adversarial coverage, i.e. the autonomous agent operates in an environment that contains threats that might stop it at any time; robocup soccer leagues style, i.e. competing with other agents for resources
- This problem is hard because: the state space is huge, thus finding an optimal solution is computationally intractable
- An exhaustive approach to the solution would be: consider all possible sequences of events (and find the one which maximizes your objective function)

From Single Agent to Multi Agent

Key differences from the single agent setting:

- The objective has changed. While a single agent tried to cover as much ground in a given time, in the multi robot setting the agent tries to collect more dirt than the adversary
- In a single agent setting we only consider the agents' plan vs a multi agent setting where we also consider the adversary plan
- In the multi agent case the adversary can alter the environment, i.e. collect dirt, which will affect the plans of your agent

Model

The given map represents a continuous workspace with known obstacles and dirt locations. At each time step t the locations of both robots is updated (via odometry measures) as well as the remaining pieces of dirt.

Suggested Approach

Our suggested solution is to aim for the piece in ”middle”, denoted as the ”tie breaker”, such that if we can get to it first, we guarantee that the number of remaining pieces of dirt on ”our side” is larger. We then split the environment in two based on both robots locations, which means that we aim to be in between the adversary and the pieces of dirt in ”our side” so that we can block him (this however is not guaranteed). We repeat this process whenever ”our side” does not have more pieces of dirt or if the adversary got to the ”tie breaker” piece first.

Suggested Approach - Example

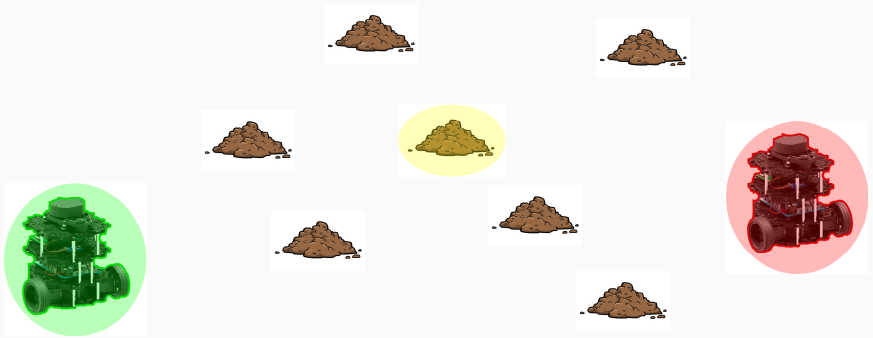


Figure 1

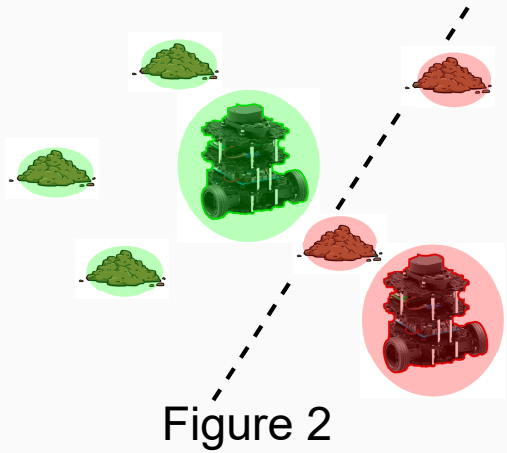


Figure 2

Our approach: Figure 1 describes an initial state where our robot (highlighted in green) identifies the ”tie-breaker” (highlighted in yellow). In Figure 2 we describe what happens if when we get to the ”tie-breaker” first. The environment is split into two, based on robots’ locations and each dirt is classified accordingly (done via simple linear SVM). We continue to collect pieces of dirt only on ”our” side as long as we have the theoretical advantage

Performance Guarantees

- **Completeness** - Our approach is complete, in each time step we return a solution
- **Soundness** - A returned solution is guaranteed to be valid
- **Optimally** - The approach is not optimal, i.e. we can ”lose” even when we had an opportunity to win
- **Anytime** - No anytime guarantees. We do not produce a different solution with more time

Performance Guarantees - Continued

- **Computational Efficiency** In each step we either find the ”tie-breaker” or classify each dirt. In any case, the complexity is $O(N_d)$, where N_d represents the number of pieces of dirt (in addition to the A^* algorithm used in move base to find the shortest path to each dirt)
- **Generality** - Our algorithm will work in any given map with any number of pieces of dirt and any number for adversaries
- **Memory consumption** - The space complexity is $O(N_d) + O(N_r)$ where N_r represents the number of robots (the complexity of A^* used in move base is $O(|V|)$ which depends on how it discretizes the map)

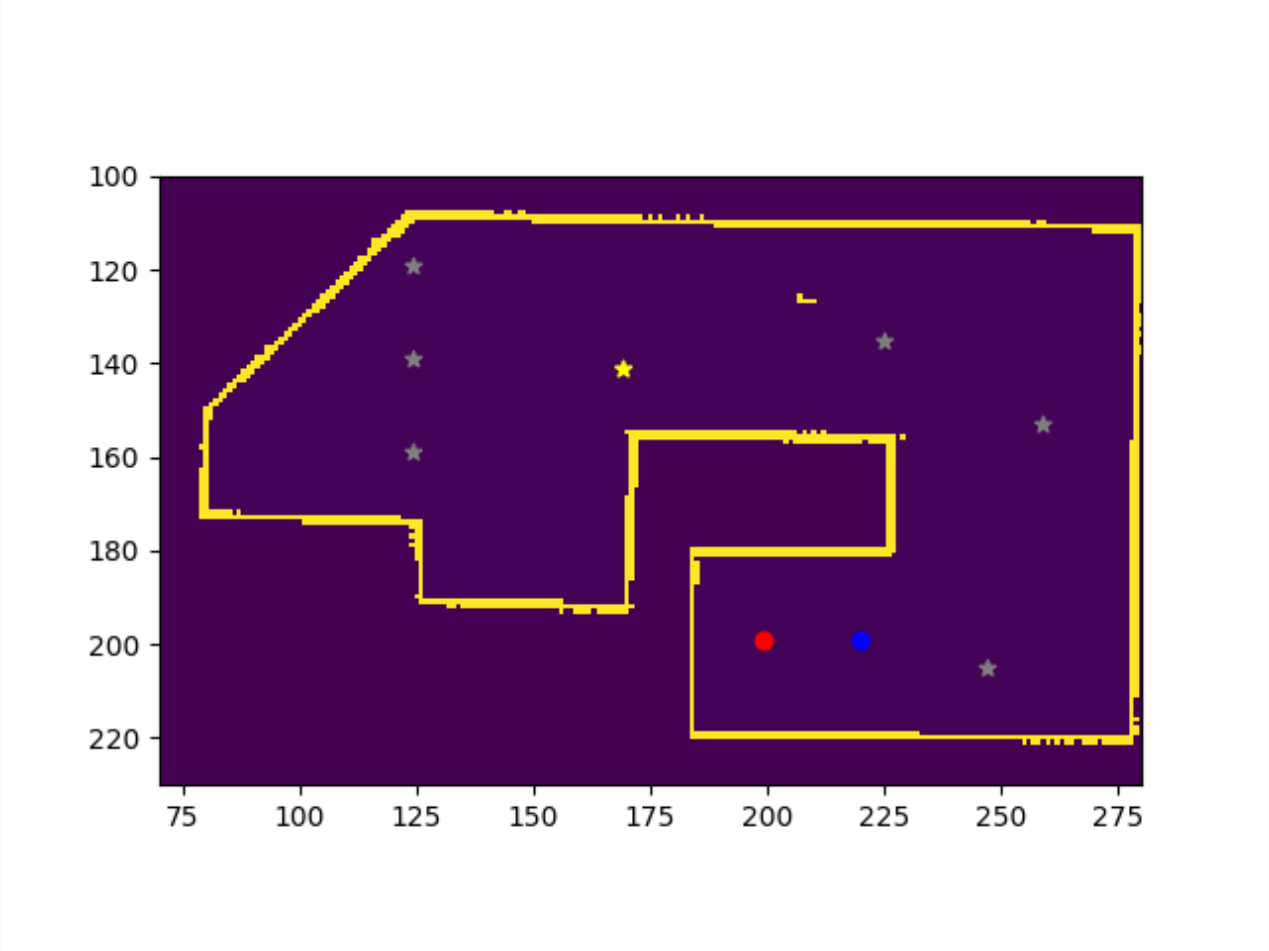
Implementation Details

In our implementation, the robot tries to collect more pieces of dirt in the following manner:

Algorithm 1 Cleaning task with adversary

```
1: Init environment
2: while environment.hasDirt() do
3:     Identify tieBreaker
4:     go to tieBreaker
5:     if collected tieBreaker then
6:         Split environment based on robots' positions ▷ Our side vs. Adversary side
7:         Classify each piece of dirt (using linear SVM)
8:         while Our side has he theoretical advantage (can guarantee a win) do
9:             collect closest dirt on our side
```

Running Example



Details from the process

- The approach we presented was the third we tried. The first approach guided our robot to the closest dirt. The second approach guided our robot based on the adversary moves. These approaches won less times in our simulations testing
- Our approach can provide guarantees for reaching to a specific piece of dirt first, however, as it is a heuristic in general, it can not guarantee to collect more pieces of dirt
- For future development, we would try to combine our approach with identifying the next moves of the adversary to better block him (currently we only go for the closest dirt on ”our side”)

Evaluation

- To evaluate our approach we randomly placed an odd number of pieces of dirt in different environments and ran the simulation vs an adversary robot
- The adversary robot we implemented tries to reach to the closest remaining dirt in each step
- In total, over 100 runs, our robot collected more pieces of dirt in 57% of the test cases

Help Requests With a Helper Agent

Problem Description

Navigating robots suffer from inherent uncertainty regarding their location due to limited sensing capabilities and computational resources. This uncertainty may cause a robot to stray from its planned course of action, and may lead it to irrecoverable situations, e.g., obstacle collision. Navigation and path-finding algorithms must take this uncertainty into consideration to avoid these situations and successfully guide the robot to its goal. In some settings it might be possible to reduce this uncertainty by utilizing another agent to provide the robot with additional information that could potentially yield a more accurate estimate of the robot's location, i.e., to provide the robot with localization information. Such information sharing operations may be costly and limited due to their high energy consumption.

Problem Description (Continued)

We want to determine when the agent will benefit the most from localization information provided by a helper agent, while taking the threshold for requesting assistance and the cost of assistance into account.

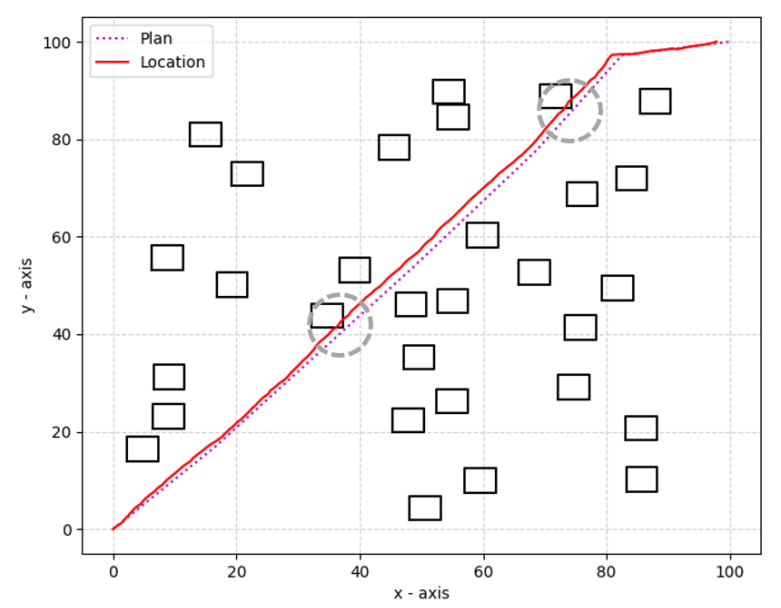


Figure 1: An agent's planned path (no collision) and real location (2 collisions) moving on the map: dotted line - planned path, red line - agent's true location.

Problem Description (continued)

Relevance to real world applications: A stationary helper agent in the form of sensor deployment[1], communication under bandwidth constraints[2], information vs. cost of communication[3].

Problem's complexity: This problem is a reduction from assignment problem which is NP-complete.

---

[1] Guohong Cao Guiling Wang and Thomas F. La Porta (2006). "Movement-Assisted Sensor Deployment". In: *IEEE TRANSACTIONS ON MOBILE COMPUTING* 5.6.

[2] Ryan J. Marcotte et al (2020). "Optimizing multi-robot communication under bandwidth constraints". In: *Autonomous Robots*.

[3] Shushman Choudhury et al (2020). "Adaptive Informative Path Planning with Multimodal Sensing". In: *Thirtieth International Conference on Automated Planning and Scheduling*.

Problem Description (continued)

An exhaustive approach to the solution: Examine every opportunity for providing information to the main agent along the intended path where the helper can supply information. The agent is then repeatedly simulated traveling along the planned path using localisation information in those spots. Compare the simulated results to all of the aid variations and choose the one with the lowest collision probability for the primary agent and the lowest cost for the helper.

From Single Agent to Multi Agent

Key differences from the single agent setting:

- **Changing cost for help** - The cost of assistance is no longer a constant figure; instead, it varies depending on the location of the help requested and the locations of both agents.
- **Heavier calculations** - Added computation of the route and time required for the helping agent to reach the primary agent to assist them.

Dimensions of Multi-Agent Systems

- **Control** - Hybrid
- **Decision making approach** - Planning
- **Communication** - Explicit communication
- **Observability** - The primary agent lacks external environment sensing capabilities and can only track its progress with odometry sensors, whereas the helper agent has sensing capabilities and can locate the primary agent when it is nearby.
- **Shared resources/interaction** - When the primary agent requests assistance, the assisting agent engages with it and provides location information.
- **Objectives/utilities** - The main agent's objective is for the to reach the goal with the lowest chance of collision, and the helper agent's objective is to assist the main agent.

Model

The map is modeled as a continues space with known obstacles containing a collision free planed path from an initial location to a goal location.

The main agent is modeled using a Gaussian distribution $x \sim N(\mu, \Sigma)$. Where x is the belief state, μ is the mean and Σ is the covariance. The mean and covariance of the initial belief are known, and the next state probability x_t linearly depends on the previous state probability x_{t-1}, the current action u_t and the process noise ϵ_t.

The helper agent is modeled as a consecutive series of locations starting from a predetermined initial location on the map.

Suggested Approach

The **inputs** to our suggested solution are the map, the main agent's planned path from its initial location to its goal (collision free), the moving agent's modeled values for the state probability function, both agents' speed and initial location, the value of the minimum threshold for the main agent to ask for localization information, and the amount of standard deviations for the Gaussian distribution.

The **output** of our proposed approach is a list of places where localization information is requested by the main agent from the helper.

Suggested Approach (Continued)

Algorithm 2 depicts our approach. The main agent's movements and location uncertainty are estimated using the intended path and *Kalman Filter* (KF)[4] (a method for linear Gaussian state estimation). Then, using the covariance ellipsis of the agent's location uncertainty (with a predetermined number of standard deviations), we evaluate whether the probability of collision within the ellipsis is greater than a predetermined threshold. In this situation, an assistance request is added to the previous time step. The request is then added to the list of final destinations where assistance will be provided by the helper, and the cost of assistance grows according to the path the helper had to travel.

[4] R. E. Kalman (1960). "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering*, pp. 35–45.

Suggested Approach (Continued)

Algorithm 2 *MovementWithHelp*

Input: Map - Map of the environment; Path - main agent's planned path from initial location to the goal; Helper- helper agent, Agent - main agent; thd - min probability threshold for help requests; std - standard deviation for Gaussian distribution.

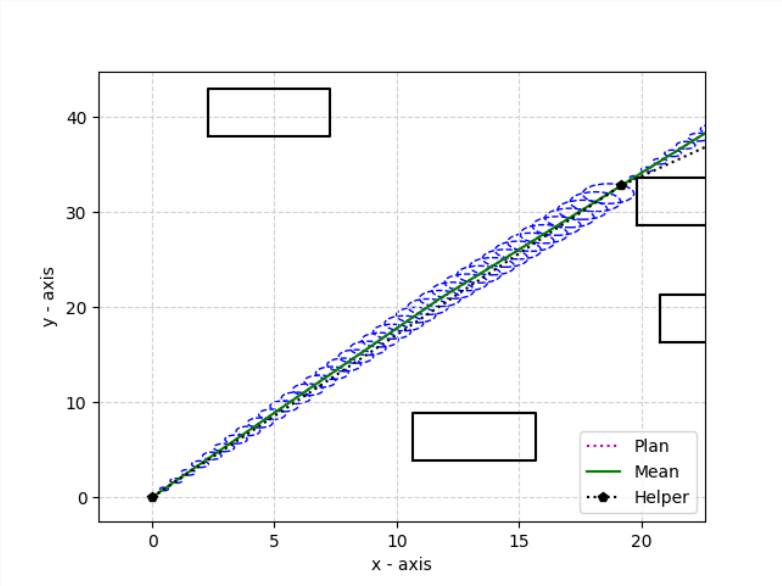
```
1:  for Step in path do                                  ▷ find where help is needed
2:      Mean,Cov ← KalmanFilter(Path, Step, Agent)
3:      ellips ← LocationEllipsis(Mean, Cov, std)
4:      Cross ← CrossSection(Map.Obstacles, ellips)
5:      ProbCollide ← P(Z ∈ Cross)                       ▷ Gaussian distribution probability
6:      if ProbCollide > thd then
7:          Requests ← PreviusStep
8:          Distance ← distance(Helper.Location, Request.Location)
9:          HelpCost ← HelpCost + Distance
10:         Helper.Location ← Request.Location
```

Implementation Details

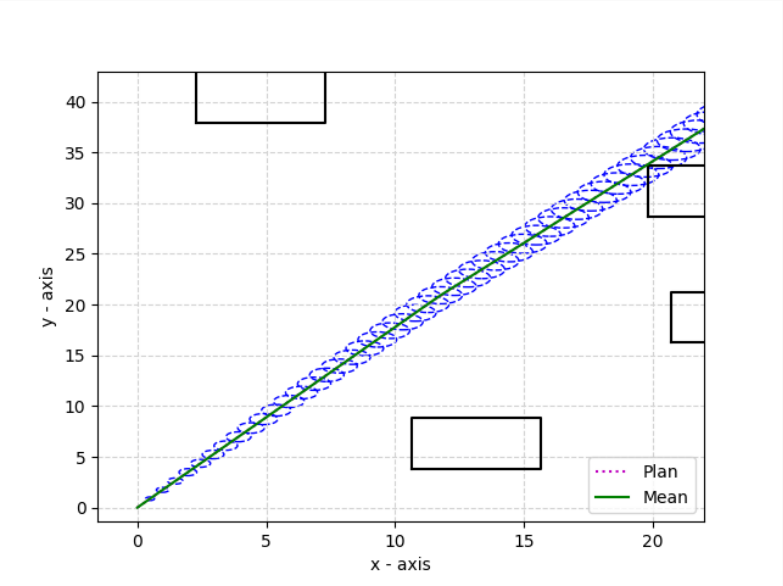
We implemented our approach using a Python environment. First, we make a map with obstacles that are placed at random. Then a *Probabilistic Roadmap* (PRM)[5] graph is constructed, with nodes indicating collision-free configurations and connecting edges representing feasible paths between them. A^* is utilized across the graph to identify the shortest path from the bottom left corner of the map to the top right corner. Then, as explained in algorithm 2, we decide where the helper agent will aid the main agent and calculate the cost of the assistance.

[5] Lydia E Kavraki et al. (1996). "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE transactions on Robotics and Automation* 12.4.

Implementation Details(Continued)



(a) With help



(b) Without help

Figure 2: Estimated location uncertainty using Kalman Filter of a robot moving from the bottom left corner to the top right corner of a map: Green - location mean, blue circles - covariance of the location uncertainty (3 standard deviations), black pentagon - locations of helper from helper agent

Performance Guarantees

- **Completeness:** Yes. The algorithm will give a list of locations for the helper to give localization information to the main agent.
- **Soundness:** Yes. If a set of locations if found then it is possible for the helper to give localization information in them to the main agent.
- **Optimally:** No. The solution does not necessarily minimize the probability of collision over the entire path and the cost of help.
- **Anytime:** Yes. The longer the amount of time given for the algorithm the more of the path the algorithm can examine for potential help requests.

Performance Guarantees (Continued)

- **Computational Efficiency:** In each time step of the main agent there are a finite amount of calculations, meaning that the computational efficiency depends only on the amount of time steps $O(m)$.
- **Generality:** The algorithm can work in any known static map with varying Start and Goal locations. Under the defined assumptions.
- **Memory consumption:** The algorithm's main memory consumption comes from the map and the planned path.

Details from the process

- We went with the second technique we tested for estimating collision probability. Initially, the size of the cross section was used to assess the likelihood of collision, but it was not representational of the true likelihood of collision.
- When the planned path passes very close to an obstacle for an extended period of time, our approach fails to adequately assist the main agent in avoiding collisions.

Evaluation

To test our method, we developed a simulation of the agents' movements to see if the main agent collided with obstacles while receiving aid in the specified positions at various assistance thresholds. We generated maps with the same size and number of obstacles, but with the obstacles scattered about the map at random. The cost of help and the likelihood of collision were then averaged across all maps using different thresholds.

Evaluation (Continued)

As shown in Figures 3 and 4, our tests show that a threshold less than 10^{-2} has no increased effect in terms of lowering the chance of collision compared to asking for help for any probability of collision, leaving it at around 25%. A threshold greater than 10^{-1} reduces the cost of assistance dramatically while increasing the likelihood of collision up to 48% same as the case where the agent does not ask for assistance at all.

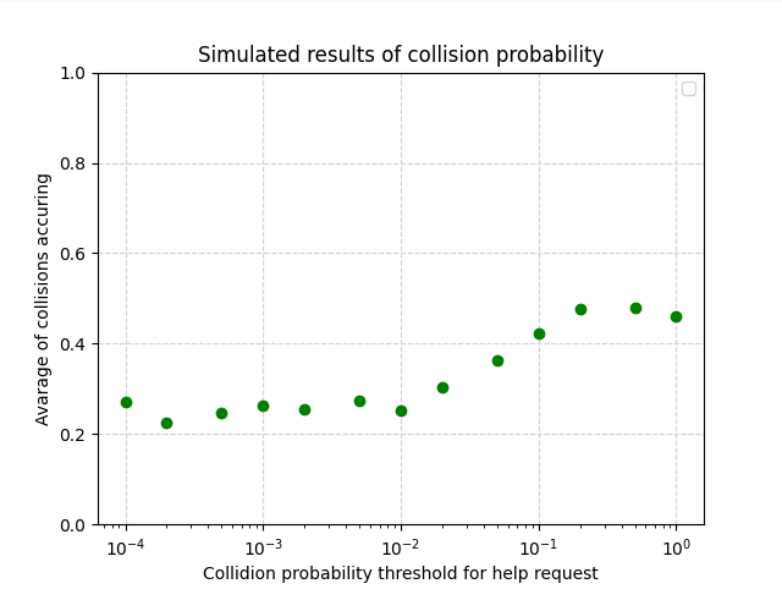


Figure 3: Collision in simulation

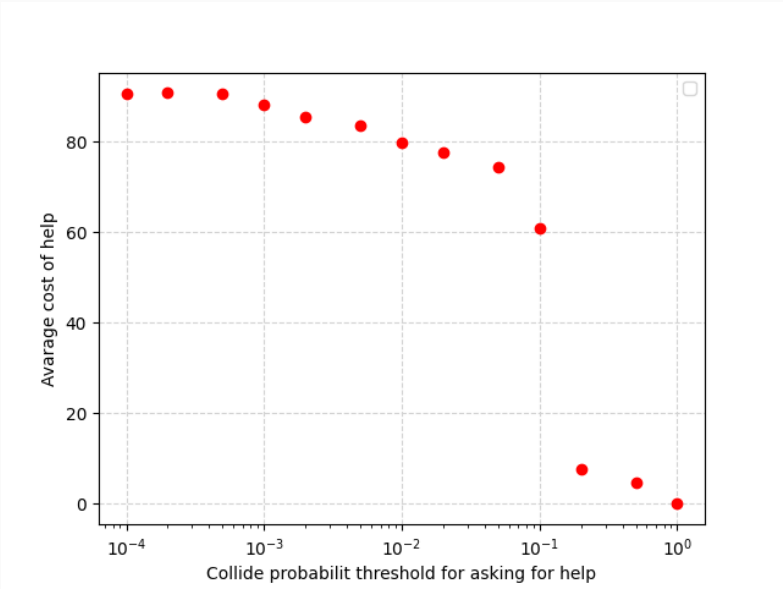


Figure 4: Help cost in simulation

Future Work

Our research goal is looking for an approach to evaluate the value of localization information given to a navigating robot, and the cost of that information.