Robot Chase

Mobile Robot Systems Mini-Project

Matthew Jackson
mj499

Michael Matthews
mm2266

***Abstract*—In this report a variety of deliberative and reactive algorithms are used to explore the chaser-runner problem with mobile robots. Controllers with coordinated path planning and target localisation are designed and evaluated over a range of speed differences.**

I. Problem definition

The problem involves two classes of agents, hereafter referred to as 'runners' and 'chasers'. The goal of the chasers is to capture all runners, by navigating to a position within a predefined capture distance of them, in a minimum amount of time. The runners have the inverse goal, aiming to avoid capture for as long as possible. The agents are subject to the following rules:

- Chasers may communicate information to each other, whilst runners may not.
- Runners are strictly faster than chasers.
- The global environment is known to all agents.
- Runners know the current position of all agents, while chasers only know runner positions when the runner is within line-of-sight (extension criterion).

The solution was extended to handle the final criterion in Sec. VI-A, with the sections prior to this assuming chasers have constant knowledge of runner positions.

In our implementation of this problem, there were three chasers and three runners tested in both an empty map with no obstacles and a complex max with a grid layout (Fig. 1). However, the controllers designed in this report were designed to generalise to differing numbers of chasers and runners, as well as different environments. Runner speed is 40% higher than chaser speed for controller comparison, with the final controller being evaluated over a range of speed differences.

II. Baseline controller

As a baseline for further analysis, a naïve solution was designed and implemented. This had purely reactive behaviour for both chasers and runners, with agents acting independently. All agents executed a Braitenberg controller, primarily to perform obstacle avoidance.

The chasers executed in two modes of operation: 'patrol' and 'chase'. While no runners were within a fixed distance of a chaser, patrol mode would be executed, with motion being determined by the Braitenberg controller alone. The sensor weights were tuned such that the agent would exhibit exploratory behaviour, searching the environment to discover runners.

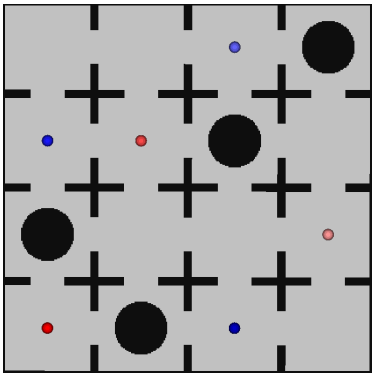

Fig. 1. Complex map — blue and red dots denote chaser and runner starting positions respectively, in a random configuration.

When a runner moved into a predefined range of a chaser, its chase mode would execute and motion would be determined by the weighted sum of the vector towards the runner and the Braitenberg controller. In order to intercept the runner along its path, the vector was set to a position in front of the robot, with the distance in front being determined by the chaser's distance from the runner.

The runners were also controlled by a Braitenberg controller, although they operated in a constant mode of behaviour. The vectors away from each chaser were weighted by an exponentially decaying function of distance from the chaser and summed with the output of the Braitenberg controller to determine motion.

III. Chaser improvements

A. Chaser coordination

From observing the performance of the baseline controller, it was clear that the runners were outperforming the chasers. Due to the speed difference between the two classes of agents, the runners were able to flee from a chaser the majority of the times one approached it, eventually escaping the range at which the chaser would pursue the runner. From this, the need for coordination between the chasers became evident, if they were to overcome the speed difference.

A centralised chaser planner was implemented in order to coordinate the chasers (Fig. 2). This allocated each of the chasers to a single runner, which it would exclusively target. Allocation was performed by computing the runner closest to any chaser, then allocating all chasers to that runner. In a

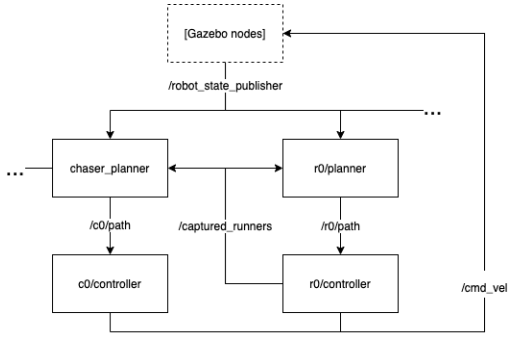

Fig. 2. ROS architecture of improved solution with centralised chaser control.

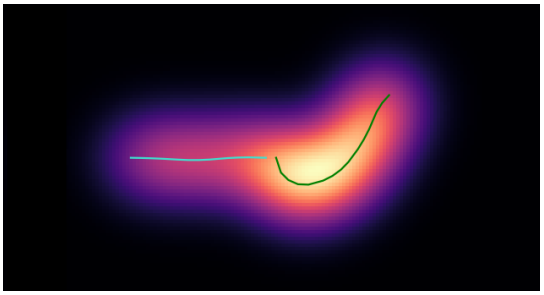

Fig. 3. Potential field from the perspective of the third chaser, considering the paths of its two teammates. The chaser will create a path that attempts to avoid the brighter areas.

configuration with more chasers, the allocation method could be adapted to target multiple runners by limiting the number of chasers allocated to each runner.

B. Chaser navigation

Observing the performance of the chasers augmented with allocation, it was observed that they would often converge onto the same path. Due to their slower speed, this allowed runners to successfully evade capture for long periods. The desired behaviour would be for the chasers to diverge and approach a runner from different sides, cutting off its routes of escape. To this end, we designed a system for chasers to cooperate.

Firstly, it was decided that, in order to facilitate cooperation, longer term navigation was required in the form of path planning with RRT*. However, rather than making this simply distance based, a potential field was also considered when planning a route. Rather than trying to minimise distance, RRT* minimises the sum of distance and the line integral through the field (1). The field was then made to be stronger around the paths of the other chasers, causing each chaser to consider paths that diverge from its teammates. This was achieved by adding a Gaussian to the field at the location of the final 10 path points of the other chasers paths (Fig. 3). Note that since the field is strictly positive, the issue of losing value by looping in a negative part of the field is avoided.

$$path = argmin_{P \in paths(\vec{s},\vec{g})}\ length(P) + \int_P F(\vec{r}) \cdot d\vec{r} \quad (1)$$

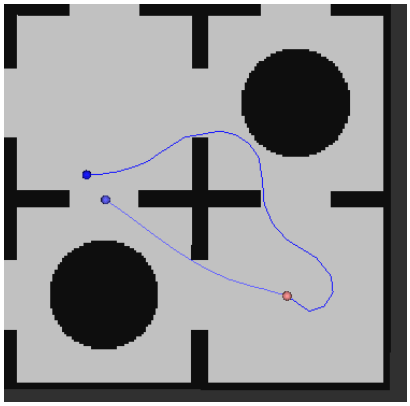

Fig. 4. Example of chaser path planning with prioritisation.

Where:

$\vec{s} =$ start position;
$\vec{g} =$ goal position;
$F =$ potential field.

In order to approximate the line integral through the potential field, the sum of the field value at each path point is taken (2). To provide a more accurate approximation, the path produced by RRT* had roughly equally spaced intermediate points added to it along the arcs between the original path points.

$$\int_P F(\vec{r}) \cdot d\vec{r} \approx \sum_{\vec{n} \in P} |\vec{n} - succ(\vec{n})| \cdot F(\vec{n}) \quad (2)$$

Where each $\vec{n}$ is a node in P and $succ$ is a function that returns the successive node in P.

It was observed that with this method that the closest chaser would often take longer routes in order to avoid the paths of its teammates, allowing the runner to escape. To fix this, we used distance to the targeted runner as the prioritisation heuristic: the chasers were ordered by their distance to the target, with each one only considering the paths of chasers that are closer to the target than it. The resulting behaviour is demonstrated in Fig. 4.

C. Evaluation

In order to quantify performance improvement compared to the baseline chaser controller, multiple iterations of the robot chase game were performed on the complex map. In this, the chaser and runner positions were randomised within the arena, then the capture time of the baseline and improved chaser controllers was measured against the baseline runner controller.

The improved chaser controller achieved 27.5% decrease in median capture time compared to the baseline controller (Table I), demonstrating a significant advantage from coordinated, deliberative control in this problem.

IV. Runner Improvements

With the improvements to the chaser architecture, the flaws in the basic runner behaviour were exaggerated. They would

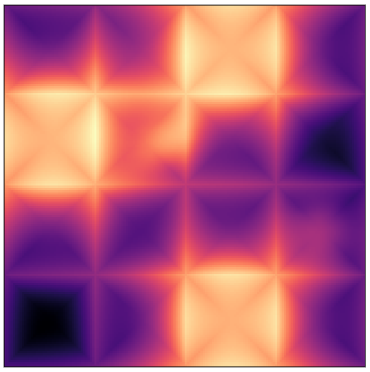

Fig. 5. Potential field of a runner, considering the configuration from Fig. 1 — brighter areas denote increased 'danger' to runners.

often allow themselves to be herded into a corner, where they could then be captured. Similar to the baseline chaser behaviour, purely reactive motion needed to be augmented with long term path planning. The path planning pipeline was implemented with two stages: candidate path generation and path evaluation.

To generate paths, exclusively distance-based RRT* was used with the small modification that there was no target position. In order to select the best path produced by RRT*, a metric of 'danger' was produced. Each chaser emitted a high level of danger around them decaying as a 2-D Gaussian. Walls emitted a low danger level as being close to them risks a runner becoming trapped against them. In order to prevent the runners clustering at the point of lowest danger, a small danger was added to each runner. Despite the runners acting independently, it was assumed that diverging would allow them to discourage chasers from clustering near them, thereby increasing capture time.

Each path is evaluated according to an approximation of the average value of its line integral through the danger field, in the same way as with the chasers. The path which minimises this value is chosen. This is subtly different to the method used in the chaser architecture, where the value in the potential field was actually used in the path generation step. By excluding the use of the potential field from this step, the runner will produce shorter paths. This decision was taken based on the observation that when in immediate danger, it is advantageous for runners to move away from it as directly as possible, using their speed advantage. Rather than winding around the areas of high danger and potentially giving the chasers enough time to surround them.

The improved runner controller was evaluated against the improved chaser using the same method described at the end of section III-C. Using the improved runner, the median time to capture increased by 26.5% (Table I), showing that long term planning is also advantageous for avoiding capture.

V. RRT* ACCELERATION

While the deliberative controllers from Sec. III and IV achieved superior performance to the initial, reactive controller, they suffered from execution speed issues due to RRT*. This led to paths being generated based on an environment state which was out-of-date by the time the paths were published. The effect of this could be reduced by greatly decreasing both the runner and chaser speeds, however this required long-running simulations and was unlikely to scale to real hardware. Therefore, a variety of acceleration techniques were implemented in order to improve the execution speed of RRT*.

TABLE I
MEDIAN BASELINE VS. IMPROVED CONTROLLER CAPTURE TIMES ON COMPLEX MAP (*speed-adjusted simulation time, seconds*)

		Chaser	
		Baseline	***Improved***
Runner	***Baseline***	215.6	156.3
	Improved	234.4	197.7

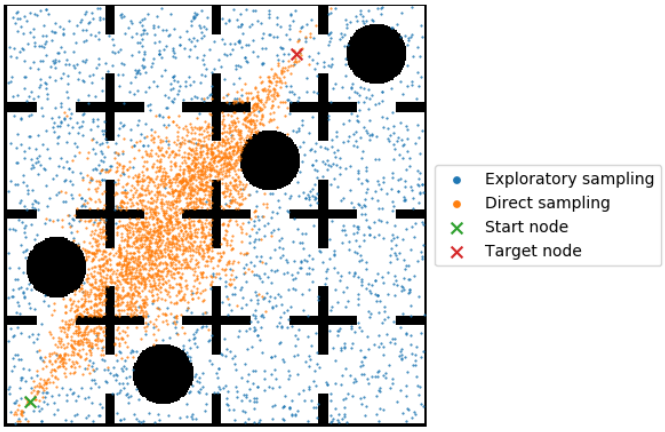

Fig. 6. RRT* Sampling. Note how the exploratory sampling doesn't choose points near the direct sampling, as these areas already have high sample density.

Firstly, a simple improvement was to store the nodes in a more efficient data structure. The world is divided into grid squares with a side length equal to that of the longest possible path between nodes. Each grid square keeps a list of the nodes in its area. This means that when considering possible edges between nodes, for instance when choosing the parent of a new node, the only candidate nodes are those in the same tile or an adjacent one. Additionally, when sampling a new node, only the subspace covered by all grid squares with at least one node or adjacent to a square with at least one node is considered.

Next, it was assumed that the majority of path points on the optimal path in a natural environment would lie close to the line between the start and goal positions. Due to this, a majority of samples were taken by first randomly selecting a position on this line, then sampling from the 1-D Gaussian distribution perpendicular to the line. The standard deviation of the Gaussian was determined by a function of distance from the midpoint of the line, producing the distribution shown in Fig.

Conversely, it would be desirable for RRT to explore less direct paths, as would be required for chaser paths to diverge. To accomplish this we make use of the grid data structure discussed earlier. If a grid square is under-populated then it is more likely to be sampled from, giving the sampling a more evenly spread coverage of the map.

Both of these sampling methods were employed to produce the distribution of points shown in Fig. 6.

VI. Localisation (extension criterion)

To satisfy the project's extension criteria, runner positions were hidden from the chasers, unless they were in line of sight of a chaser. Since the chasers could communicate, only a single chaser required line of sight for all chasers to know the runner's true location. It was also assumed that the chasers had no information about the runners' motion when they were out of sight, other than the maximum speed the runners were capable of moving.

A. Localisation method

In order to estimate the position of runners, Monte Carlo Localization was performed using a particle filter. When the environment is initialised, the particles spawn uniformly within the arena, outside of any obstacles. Each iteration of the chaser controller, the particle positions update using a random walk up to the runner's maximum speed, since the true runner motion is unknown. If a particle enters the line of sight of a chaser or an invalid position (outside of the arena or colliding with an obstacle) the particle is killed and a new particle spawns at the position of an existing particle. When a runner is observed, all particles are translated to the runner's true position.

B. Path planning with particle filters

When performing chaser allocation with particles, the initial solution was to prioritise runners whose true position was known, then find minimum chaser-runner distance as before. However, this led to chasers reallocating to faraway runners when a runner they were pursuing moved behind an obstacle. An alternative solution which resolves this was to use the centroid of each particle cloud as the runner's position, however this led to runners with wide particle clouds being targeted when the centroid was near a chaser.

In order to solve both of these issues, the worst-case distance of the runner from each chaser was used for allocation, by finding the maximum distance of any particle in the cloud from each chaser. This allowed runners with dense particle clouds (whose true position was known to be within a small area) to be targeted when near chasers, while wide particle clouds were unlikely to be chosen.

If a runner whose true position was unknown was targeted, each chaser would select the closest position in the particle cloud as its goal, in order to reduce the distribution of possible runner positions. When combined with the coordinated path planning from Sec. III-B, this led to systematic searching behaviour, as the chasers would diverge and eliminate possible runner positions until the runner was located.

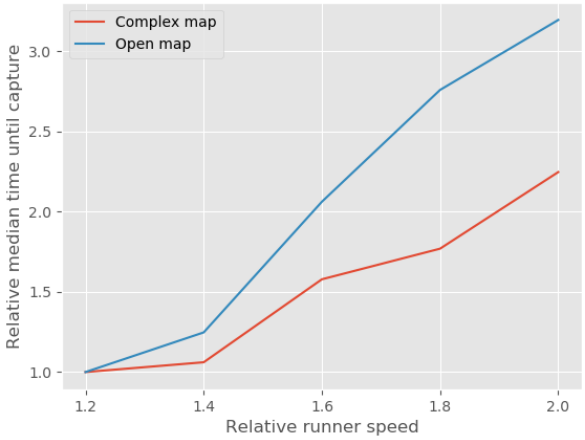

Fig. 7. Comparison of capture time vs runner speed in both the open and complex world environments. The y axis is normalised to the median time taken when relative speed = 1.2.

VII. Further evaluation

The improved controllers with localisation were evaluated over a range of runner-chaser speed differences, in both the open and complex environments (Fig. 7). As expected, the capture time rises as the runner's speed advantage is increased. However, the rate of increase is greater in the open environment, compared to the complex environment. Observing games in the complex environment, it was clear that the majority of captures occurred when a runner moved into the corner of a grid square and had its escape routes blocked by chasers. Since this only occurred on the arena corners in the open environment, the runners were able to evade capture for longer periods. Increasing the number of chasers in the game may allow runners escape routes to be removed when it is in open space, improving performance in the open environment.

VIII. Conclusion

Deliberative controllers for chasers and runners, using a combination of RRT* and potential field methods, were designed and implemented. A particle filter was implemented to perform runner localisation, thereby satisfying the extension criterion. Compared to their baselines, final chaser controller achieved a 15.7% decrease in capture time against the final runner controller, whilst the final runner controller achieved a 26.5% increase in capture time against the final chaser controller.

Both team members contributed equally to this report. Approximately, Matthew Jackson implemented the ROS architecture, simulation environment and runner localisation, while Michael Matthews implemented the RRT and potential field algorithms, although both members contributed in part to all components.*