A* PATH PLANNING ALGORITHM FOR OBSTACLE AVOIDANCE APPLIED TO SINGLE AGENT INDIRECT HERDING

By

MAX L. GREENE

ABSTRACT

Certain network systems are comprised of agents with uncertain nonlinear dynamics that are indirectly controlled and herded by the influence of other agents, during which collision and interference between agents may occur. In some circumstances, the uncontrollable (target) agents and controllable (herder) agents should not collide or interfere, such as in munition collection or arrangement.

This thesis evaluates the implementation of an A* path planning algorithm used to direct one target agent toward a goal location while navigating around other uncontrolled agents (obstacles). A heuristic approach is developed for the A* algorithm that enables obstacle avoidance and generates the least cost path to the desired goal location. It is possible to implement an A* algorithm to plan a path for this specific single agent herding problem. While this method may be implemented to herd other objects with uncertain nonlinear dynamics, it requires system-dependent tuning and reliable control of the target agent.

Literature Review

The A* search algorithm is commonly used in robotics path planning problems to identify a path between nodes. A* does not account for real-time obstacle avoidance or motion of objects [1]. When the A* algorithm is implemented, the position of objects is not expected to change. In [2], the A* algorithm is tested for minimization of execution time and traveling time. The execution and traveling times were found to vary when using an A* algorithm for path planning in a six degree of freedom stationary robot. The computation time and execution time for A* are not always faster than other methods. Other methods were evaluated, such as D* (Dynamic A*). The D* algorithm was not selected since the obstacles were assumed to be stationary. Due to its ubiquity in mobile robotics, the A* algorithm was selected as the path planning method.

Bezier curves are commonly used in robotics path planning applications for mobile robots. In [3], a sequence of cubic Bezier curves used to create a smooth path from a user drawing on a touchpad as input. In [4], cubic Bezier curve paths are generated for two-wheeled mobile robots to push a ball toward the desired location using the initial heading, final heading, and terminal velocity of the robot. Bezier curves are also commonly used in robotic obstacle avoidance. A Bezier curve is used to adhere closely to the discrete points generated by the A* algorithm. The Bezier curve is of $n-1$ order, where n is the number of nodes in the path. For simplicity, one Bezier curve along all discrete points determined by the A* algorithm is used for this method. The Bezier curve can be used to create a first-order continuous path.

The single-agent herding of multiple targets is possible as shown in [5]. There has not been a path assigned to the target agent that guides it from its start to finish location. This method explores the feasibility and performance of applying an A* algorithm path planner with a

Bezier curve to decrease interference between the controlled target agent and the uncontrolled target agents.

Introduction

Target agents can be guided toward a goal location by a herding agent as shown in Figure 1. In some cases, the desired trajectory may interfere or cause a collision between a target agent and untargeted agent, as shown in Figure 2. The method described in [5] lacks a means to decrease interference between the herding agent, target agent, and untargeted agents. A common approach to path planning for mobile robots is the A* algorithm. The A* algorithm is commonly used in robotics for mobile robots to maneuver from a starting position to a final position while avoiding impassable obstacles. If the untargeted agents are treated as obstacles, then the A* algorithm can be used to generate a path for a targeted agent to minimize interference from the herding agent.

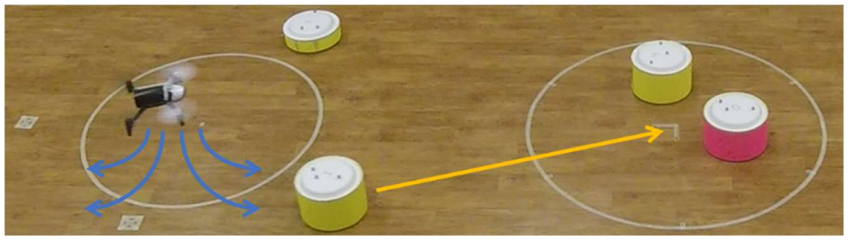

Figure 1: The blue arrows illustrate the airflow that will cause the closest yellow paper target to move along the trajectory (described by the yellow line).

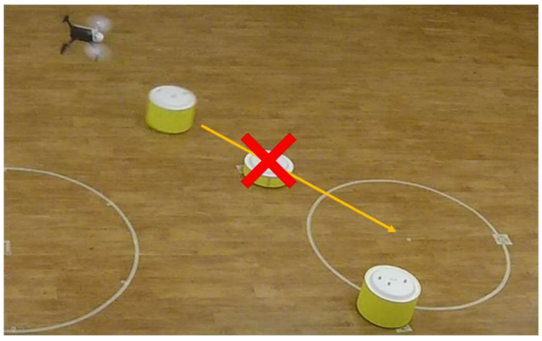

Figure 2: This figure illustrates the problem that the target agent's trajectory may intersect the position of the other paper target.

The A* algorithm is driven by a heuristic cost function, which is the sum of the distance from the final location and the heuristic function. The cost map is a set of nodes for which the cost function has been performed. The A* algorithm is performed using the cost map. This approach generates a sequence of points for the target agent to follow that is the least cost path. A Bezier curve is generated using coordinates of the nodes solved for by the A* algorithm. Coordinates on the Bezier curve are the desired path.

This technique aims to develop a least-cost path for the target agent to follow using an A* algorithm and a Bezier curve. This technique will attempt to decrease interference and decrease collision frequency between the herding agent and untargeted agents.

Problem Development

In the testing phase of [5], it was noted that the herding agent unintentionally moved the untargeted agents while moving the targeted agent toward the goal position. In this test, there was no consequence for moving the untargeted agents. There could be circumstances in which the obstacle must be moved around a set of obstacles or avoid an uncontrolled agent. In such circumstances, the herder could likely not travel in a straight line from the initial position to the goal location. A different path must be generated to mitigate interference with uncontrolled agents.

One of the more common path planning methods is the use of the A* search algorithm. The A* algorithm is commonly used in robotics applications to find the shortest distance path between an initial position and goal position while navigating around obstacles. For this purpose, the A* algorithm is modified with a unique heuristic that finds the least-cost path to navigate around the untargeted agents while traveling over a short path.

A Bezier curve is created to pass through the points generated by the A* algorithm. The Bezier curve creates a first and second order continuous curve for the target to follow. A series of cubic Bezier curves could be used to more closely adhere to the path generated by the A* algorithm. The latter method could be used to create a more geometrically complex path that cannot be represented by a single Bezier curve.

This method is tested using the same setup as in [5], but the goal location of the target agent will be continuously updated along the Bezier curve.

Cost

The A* algorithm is used to determine the least-cost neighbor node. Generally, the cost for each node is determined by $f(n) = g(n) + h(n)$, where $f(n)$ is the cost of the next node in the path sequence, $g(n)$ is the portion of the cost dictated by the distance from the goal position, and $h(n)$ is the portion of the cost dictated by the heuristic function. In this proposal, the heuristic is designed as

$$h(n) = \left| \sum_{m=1}^{n_{obs}} \left[f(n-1) \times \frac{n_{total}}{k_{weight} \times (d_{obst}(n,m) - d_{corr})} \right] \right| \quad (1)$$

In (1), each obstacle is included through the $\sum_{m=1}^{n_{obs}}$ term. The cost of the previous visited node is $f(n-1)$, n_{total} is the total number of nodes in the cost map (e.g., if the map is 30 units wide by 60 unit long, then n_{total} would equal 1800), and k_{weigh} is a constant that dictates how the distance from the obstacle affects the controlled target's path. The k_{weight} of this cost function is tuned by running simulations with expected system parameters (n_{total}, k_{weight},d_{corr}). A larger k_{weigh} decreases the importance of interference on the path. A smaller k_{weigh} increases the importance of interference on the path. Also in (1), $d_{obst}(n,m)$ is the distance from the

evaluated obstacle to the node being evaluated, and d_{corr} defines a radius from the obstacle from which the controlled object should not pass. The ability of the heuristic function to raise the cost around obstacles is depicted in Figure 3.

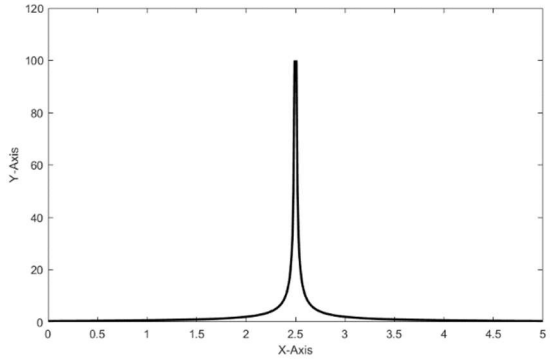

Figure 3: This plot illustrates how the heuristic raises the cost of the nodes around the obstacles. In circumstances where alternative paths may be expensive, the algorithm may force the path to go within this defined radius. Such a path would still be the least-cost, but would interfere with an uncontrolled target. In this situation, the path is not further modified and will travel through an obstacle. To avoid a collision in this case, the dimensions of the cost map must be increased so that the target agent can travel around the obstacles.

Bezier Curve

A Bezier curve is developed from the A* algorithm. The least-cost path is used as the control points in the Bezier curve. The curve is a set of two parametric equations shown by (2). The control points in the curve are the coordinates of the nodes from the A* algorithm.

$$\begin{bmatrix} B_{n,x}(t) \\ B_{n,y}(t) \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} x_i \left[\binom{n}{i} t^i (1-t)^{n-i} \right] \\ \sum_{i=0}^{n} y_i \left[\binom{n}{i} t^i (1-t)^{n-i} \right] \end{bmatrix} \tag{2}$$

$\binom{n}{i} t^i (1-t)^{n-i}$ is a Bernstein basis polynomial of degree n, and the parameterized curves are a function of t, where $t \in [0,\ 1]$.

Simulation

The A* algorithm was tested through simulation in MATLAB. The path illustrated in Figure 4 is tested such that the heuristic is equal to zero. Figure 4 shows a path without object avoidance. This produces the shortest path between the starting and goal location. The path in Figure 4 is the shortest path. Note that the path travels directly through a set of three obstacles, which is undesired. In Figure 4, the least cost path is not a straight line. The path is not a straight line because the A* algorithm can only travel at 45° diagonals. This limitation is why the path in Figure 4 is not a straight line between the start and goal position.

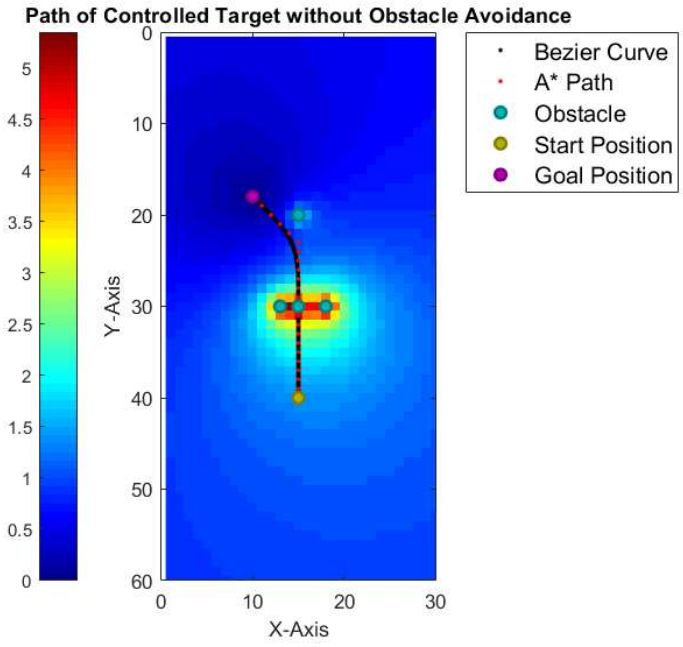

Figure 4: Controlled target path through a 2D plane with no obstacles.

The A* algorithm is tested with the heuristic in Figure 5. The results of the fully implemented algorithm are pictured in Figure 5. The A* algorithm travels around all four

obstacles and to the goal location. The path generated in Figure 5 proves that the cost model in the A* algorithm is valid. This algorithm guides the target agent to the goal position while traveling around obstacles with a least-cost path.

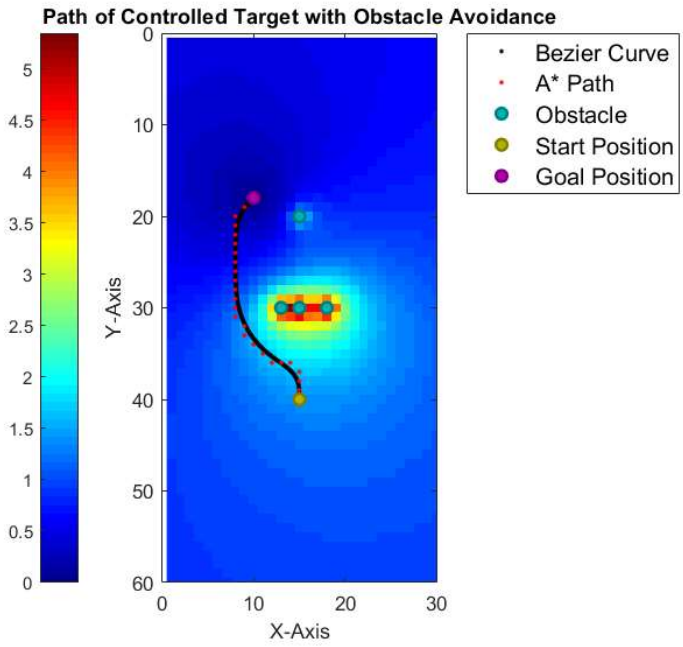

Figure 5: Controlled target path through a 2D plane with four obstacles.

Experimentation

The path must be tested to see if it can be implemented on a single agent herding problem. The experiment is tested using the hardware specified in Figure 6. The ROS Core computer collects data from the Intel mini PC, motion capture system, and a laptop. The laptop executes the A* algorithm and provides the ROS Core computer with the desired positions. The ROS Core computer uses the desired path data and the current state of the Parrot Bebop drone, controlled target, and uncontrolled targets to implement the pathfinder.

MATLAB is run externally on a laptop connected to the ROS Core computer. The laptop calculates the path using the A* algorithm. Latency is not an issue in this experiment because the laptop will publish all path to the ROS Core before the ROS Core executes the herding algorithm. The ROS Core will publish commands to the Intel NUC Mini PC using the herding algorithm created to test herding in [5].

An Optitrack Motion Capture system was used to collect the position state for the drone and paper targets. Initially, the motion capture will publish the paper target positions to the ROS Core computer, which will publish that information to a laptop to execute the path planning algorithm.

The paper targets will be moved by the airflow from the drone. Tracking by the motion capture system will capture the final position state to see if the untargeted paper agents were moved by the drone airflow.

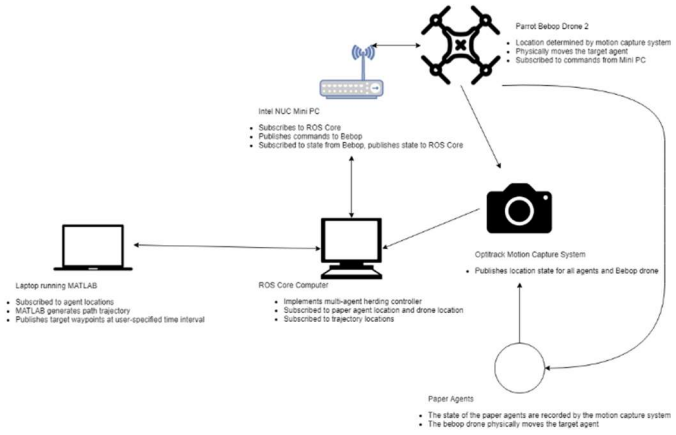

Figure 6: Flow diagram for the experimental test setup.

In the experiment, one target agent is tasked to travel toward the goal location while navigating around three obstacles. The Optitrack cameras detects the position of the three obstacles and the target agent. The user specifies the desired time for the target agent to reach the

goal position. The test ends when the target agent is within 0.5 meters of the goal location. The A* path planning algorithm is executed and the desired target agent path is determined. The ROS Core computer runs the herding algorithm that enables the drone to herd the target agent toward the dynamic goal location.

The error in the experiment is calculated with root mean squared (RMS) error. RMS error is taken by summing the actual target agent distance from the desired target agent location, taking the average of that sum, and then taking the square root of that quantity.

Figure 7 displays the results of the first experiment. The desired time is 15 seconds. While the actual target agent location reached within 0.5 meters of the goal positions, it did reach the goal location ahead of the desired time. The drone did not have control of the target agent for the duration of the experiment.

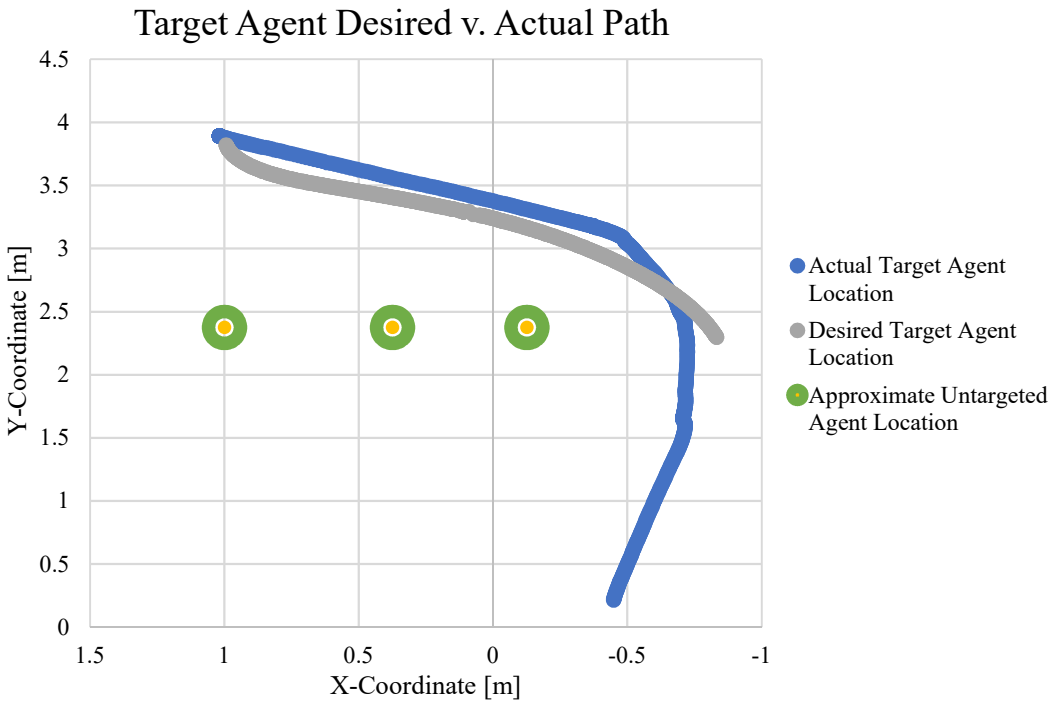

Figure 7: Plot of the actual target agent location compared to the desired target agent location.

The RMS error for this experiment is 1.29 meters. This error is too high to confirm successful implementation of this test. Tuning the low-level control of the herding algorithm and decreasing the desired time may yield better results. This experiment proves that the system properties impact the ability to follow the desired path. Such properties for this experiment include the geometry of the target agent, the strength of actuation (air thrust magnitude), and the drone height relative to the ground.

CONCLUSION

It is possible to implement a custom A* path planning algorithm to single agent indirect herding. Simulated trajectories of the path planner show that the A* algorithm used in this thesis is a viable option for obstacle avoidance. Experimental results showed that the herding agent guided the target agent around the untargeted agents toward the goal location, though the target agent did not follow the path with great accuracy. The herding agent could not be reliably controlled to herd the target agent. Creating a stronger low-level controller over the herding agent may give the herding agent more control over the target agent's position. Applying a stronger low-level controller that incorporates the herding agent's dynamics may allow the target agent to move along any generated path accurately. While an A* algorithm can reliably generate a path to limit interference between agents, the herding controller must incorporate herding agent dynamics to be implemented successfully.

REFERENCES

[1] B. Balasubramanian, "A New Guidance Trajectory Generation Algorithm for Unmanned Systems Incorporating Vehicle Dynamics and Constraints," Blacksburg, VA, 2010.

[2] F. J. Abu-Dakka, F. Rubio, F. Valero and V. Mata, "Evolutionary indirect approach to solving trajectory planning problem for industrial robots operating in workspaces with obstacles," *European Journal of Mechanics - A/Solids,* vol. 42, no. November-December, pp. 210-218, 2013.

[3] J.-H. Hwang, R. Arkin and D.-S. Kwon, "Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, 2003.

[4] K. Jolly, R. S. Kumar and R. Vijayakumar, "A Bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits," *Robotics and Autonomous Systems,* vol. 57, no. 1, pp. 23-33, 2009.

[5] R. A. Licitra, Z. I. Bell, E. A. Doucette and W. E. Dixon, "Single Agent Indirect Herding of Multiple Targets: A Switched Adaptive Control Approach," *IEEE Control Systems Letters,* vol. 2, no. 1, pp. 127-132, 2018.