Multi-Agent Pursuit of a Rogue Agent in an Urban Setting: Catching Batman

Soumya Chakraborty, Sumit Mukherjee, Andrew Winn

December 5, 2012

A final report submitted for the course of
Distributed Systems and Sensors
Fall Semester 2012

Contents

1 Introduction **1**

2 Theory **2**

3 Road Network **2**

4 Stationary Sensor Network **3**

5 Pursuer **4**

6 Evader **6**

7 Action Handling **7**

8 Simulation Specifications **7**

9 Results **7**

10 Conclusion **12**

11 Additional Figures **14**

List of Figures

1 Proposed Problem Setup 2
2 Stationary Sensor Network Functioning Flowchart 4
3 Pursuer Pinging Sensor Network Flowchart 5
4 Pursue Algorithm 1 Flowchart 5
5 Pursue Algorithm 2 Flowchart 6
6 (a) Algorithm 1: Average Response Time vs Number Of pursuers and Normal Cars (b) Algorithm 2: Average Response Time vs Number Of pursuers and Normal Cars 8
7 Algorithm 1: Sensor Idle Time vs Number Of pursuers and Normal Cars 8
8 Algorithm 1: Sensor Idle Time vs Number Of pursuers and Normal Cars 8
9 Algorithm 1: Max Messages Sent by a Sensor vs Number Of pursuers and Normal Cars 10
10 Algorithm 2: Max Messages Sent by a Sensor vs Number Of pursuers and Normal Cars 10
11 Algorithm 1: Number of Trials Converged vs Number Of pursuers and Normal Cars 10
12 Algorithm 2: Number of Trials Converged vs Number Of pursuers and Normal Cars 10
13 Algorithm 1: Time Steps to Convergence vs Number Of pursuers and Normal Cars 11
14 Algorithm 2: Time Steps to Convergence vs Number Of pursuers and Normal Cars 11
15 Algorithm 1: Worst Response Time vs Number Of pursuers and Normal Cars 11
16 Algorithm 2: Worst Response Time vs Number Of pursuers and Normal Cars 11
17 Algorithm 1: Time Steps to Converge vs Number Of pursuers and Ratio of Evader to Pursuer Speed 12
18 Algorithm 2: Time Steps to Converge vs Number Of pursuers and Ratio of Evader to Pursuer Speed 12
19 Algorithm 1: Max Messages per Time Slot vs Number Of pursuers and Normal Cars 14
20 Algorithm 2: Max Messages per Time Slot vs Number Of pursuers and Normal Cars 14
21 Algorithm 1: Min Idle Time vs Number Of pursuers and Normal Cars 14
22 Algorithm 2: Min Idle Time vs Number Of pursuers and Normal Cars 14
23 Algorithm 1: Number of Messages vs Number Of pursuers and Normal Cars 15
24 Algorithm 2: Number of Messages vs Number Of pursuers and Normal Cars 15

25 Algorithm 1: Total Number of Messages vs Number Of pursuers and Normal Cars 15
26 Algorithm 2: Total Number of Messages vs Number Of pursuers and Normal Cars 15
27 Algorithm 1: Number of Converged Trials vs Number Of pursuers and Ratio of Evader to Pursuer Speed 16
28 Algorithm 2: Number of Converge-ed Trials vs Number Of pursuers and Ratio of Evader to Pursuer Speed 16

1 Introduction

Consider an urban environment where there are many agents that move along a grid-like network of roads to travel from some source location to some destination location. These cars are constrained to obey traffic laws, that is, their speed is bounded by the speed limit, and they can only travel through intersections In addition, there is a static network of sensors at each intersection that record when a car passes through and what its velocity is at that time. Our problem is that of pursuing a rogue agent traveling through the city with a set of drone vehicles, which can be thought of as the police. These drones can communicate with the stationary network and can be treated as mobile nodes within the network. We assume that the stationary nodes are also capable of distinguishing the rogue agent from the rest of the vehicles, and can pass this information to the pursuing agents. The pursuing agents are further constrained to maintain safety, that is, they will maintain some safe distance from other vehicles and will not unsafely pass through red lights, whereas the rogue agent is not required to follow these constraints. A pictorial example of this problem setup is given in Figure 1.

In order to address the problem, we have the stationary network collect data concerning the current location and movement direction of the rogue agent, as well as current congestion levels on different sections of road, and make this information available to the pursuing agents. We then referred various sources on multi-agent pursuit, e.g. [1] [4] [5] [6], to determine a strategy for the rogue agent to evade and the pursuing agents to chase given the constraints. We then examine how different algorithms and protocols affect the efficacy of the pursuit algorithm, and draw conclusions regarding the network configuration that produces the best results.

A wireless sensor network demands implementation of protocols and algorithms that make effective use of limited resources and counter the challenges posed by their operating environment. The network in our case comprises many static nodes and a few mobile nodes. A similar structure is examined in [3], and their results guided our network design. The major limitation was the limited battery power for the stationary nodes. In wireless sensor networks, communications takes the most amount of energy, so it is very essential to strike the right balance between data processing and transmission of raw data. We assumed that sensor nodes have the capability to do some amount of signal processing before they forward or disseminate the data. The topology is such that one can expect that at a particular time only a few stationary sensor will be in the vicinity of a mobile sensor. This boosts the case for giving control on part of the mobile nodes to initiate and fetch information from the stationary nodes. The degree of mobility of the nodes is another factor that determined the efficiency of pursuit algorithms used. We have tested our algorithms by simulating the system in MATLAB while varying different parameters of the simulation, such as the starting location of the agents, the number of pursuers, the density of

vehicles, the communication speed and range of sensors, etc.

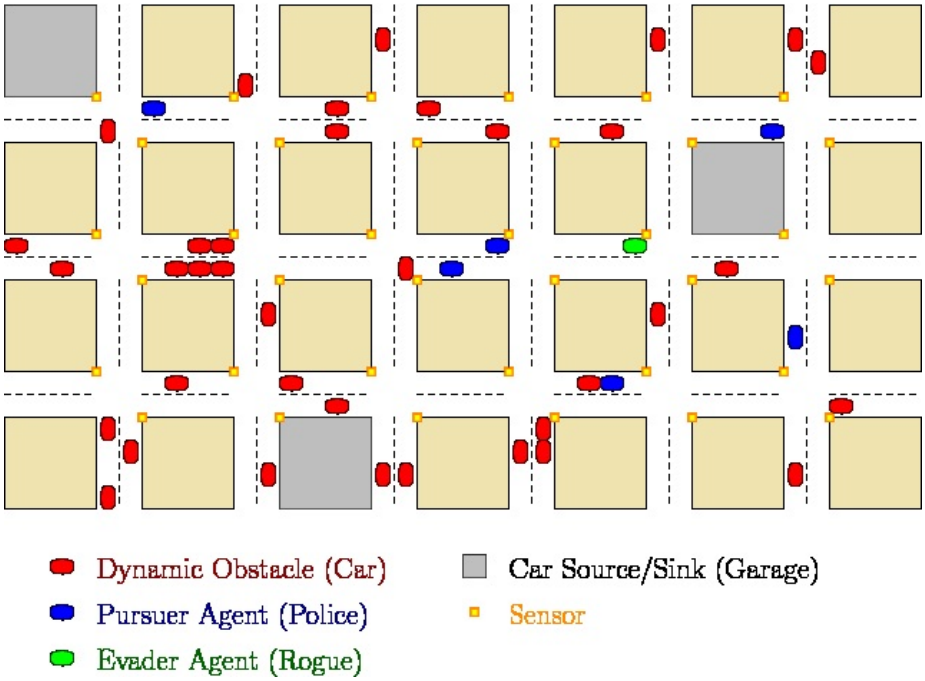


Figure 1: Proposed Problem Setup

2 Theory

The Chase begins with the all the pursuer cars having knowledge of where the evader (batman) was spotted. The pursuer car pings the sensor node asking for the current location of batman (intersection batman is headed to) after a finite interval of time in case it does not have the information on its own. The sensor returns the data if its information is updated else it forwards it to its neighbor and so on till it gets an updated information about the evader. The pursuer uses the updated information to choose the least congested path on its way towards batman.

3 Road Network

The road network consists of a set of interconnected roads that spans the entire city. The roads meet at intersections at the end of each block. The road network

has been configured to have block length, number of forward lanes, number of backward lanes, lane width, shoulder width, intersection radius and car length in parameterized form. This enables to simulate the city roads in as realistic way as possible.

4 Stationary Sensor Network

Sensors are located at the center of each intersection. Each sensor (except the one at edges) has four neighbors one on left, one on right, one up and one below it. By neighbors, we mean the sensors which the sensor can communicate with directly. Sensors can see cars going any direction from the intersection and store that information. They can differentiate between the evader car and the pursuer car. Sensors see the direction the evader is heading to and can directly return the intersection it is headed to. In practical terms it is basically coordinates of the intersection; however for simplicity, we consider the index of the neighbor.

When a pursuer car requests any sensor for current location of the evader, sensor sees the information it has about the evader's last location and matches its time stamp for validity. If the information is valid, it directly return the data to the pursuer. It is also possible that multiple pursuers are requesting the data from a single sensor. We handle this situation by giving power on behalf of the sensor to maintain a list of pursuers it need to return back the information. This way, multiple forwarding requests is avoided. In the event that the sensor node being pinged does not have valid information about the batman, it updates its own information with that of the pursuer and forwards the request to one of its neighbor based on which on is closely located from the intersection batman was last headed to. This process continues till an updated information about evader's location is found by one of the nodes which then returns it back to the sensor which initially started the forwarding request in the minimum number of hops. The setup of our problem is such that the pursuer that requested a sensor for information could at most move to one of the neighbors by the time the information comes back; which is a reasonable assumption since on an average the communication in a network happens faster than the vehicle movement. To account for this, information received by the source sensor is also forwarded to its neighbors.

An interesting situation arises when the source node that initiated the request becomes the node to which batman is heading towards. The sensor node informs the pursuer of the same and the pursuer head towards that intersection, seeing which the evader makes a U-turn. To incorporate this the sensor node waits for a finite time to see whether the evader indeed crossed the intersection or not and updates the pursuer accordingly. The flowchart for the network operation is shown in figure 2.

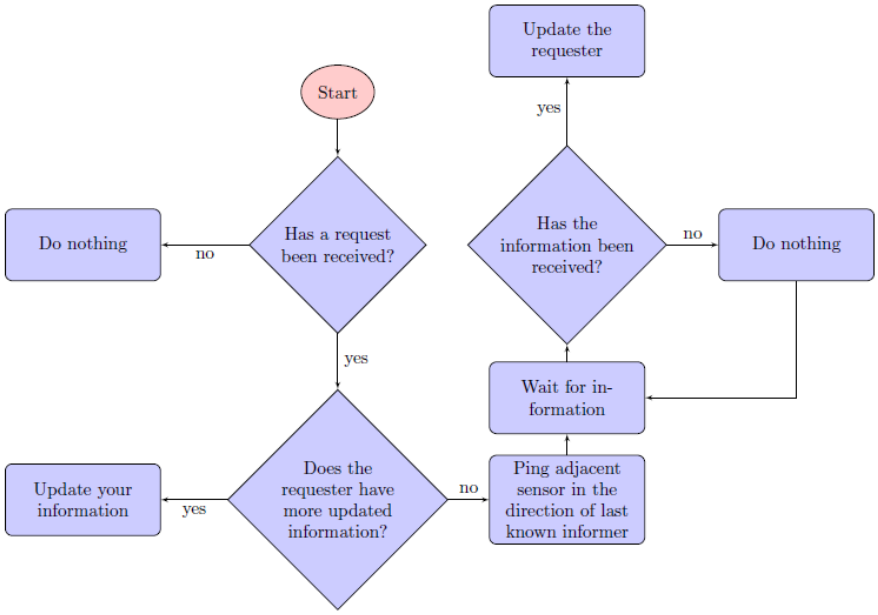


Figure 2: Stationary Sensor Network Functioning Flowchart

5 Pursuer

Pursuer pursuits the evader in two different ways. In the simplistic model, the pursuer keeps moving until it reaches an intersection. When at an intersection, pursuer chooses the shortest path leading to the evader. It is possible that the pursuer can have multiple options or a single option depending upon whether it is in the middle or edge of the city area. The pursuer chooses the path with least congestion of all the paths it can take at that time. In order to always have a correct estimate of where the evader is, the pursuer keeps checking whether the difference between the current time and the time of last known location of evader is less than a finite time called del-bat. This finite time, del-bat is roughly the time evader takes to cross from one intersection to the another. In case the difference of two time is greater than the del-bat, the pursuer connects with the sensor it is closest to unless it is not already connected and pings for updated information about the evader. The flowchart for this is seen in figure 4. In a slightly complex variant of this model, the pursuer on arriving at an intersection sees if all the shortest paths leading to the evader has already a pursuer on the way. If another pursuer is present on all of those paths, the pursuer at the intersection either chooses an alternate shortest path if there is one or chooses to move on the road with least congestion. If there are roads that does not already have a pursuer, then the pursuer at the intersection chooses the one among them which is least congested. The flowchart for this is seen in figure 5.

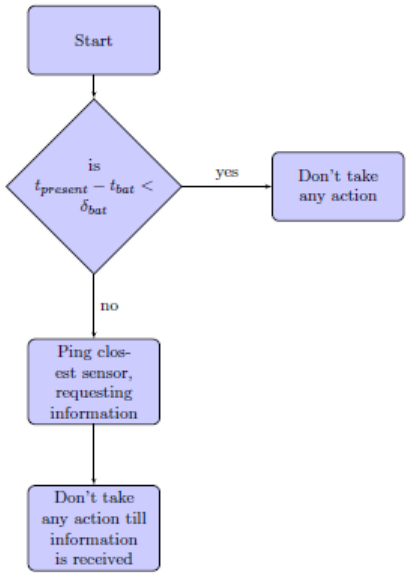


Figure 3: Pursuer Pinging Sensor Network Flowchart

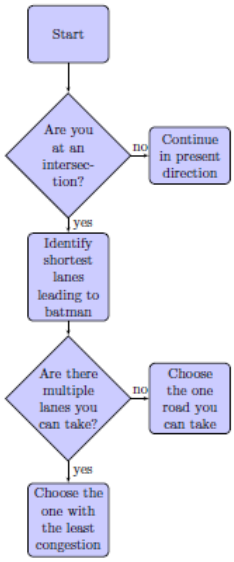


Figure 4: Pursue Algorithm 1 Flowchart

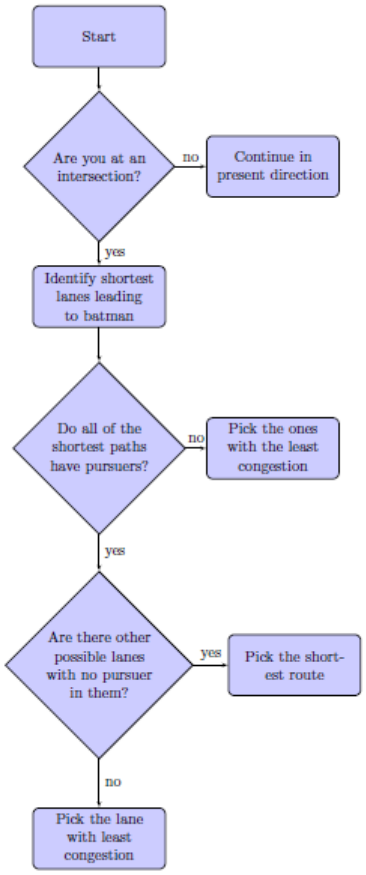


Figure 5: Pursue Algorithm 2 Flowchart

6 Evader

In general, the evader keeps moving like the pursuer and normal cars until it reaches an intersection. The evader on reaching an intersection chooses a path that does not have a pursuer and if there are multiple options it randomly chooses a least congested path. In order to give power to the evader, it has been given the capability to make a u-turn on seeing a pursuer in front. This ensures that one pursuer car on its own cannot bust the evader even if it has been cornered to the edge or elsewhere.

7 Action Handling

For algorithmic and computation simplicity, we implement a discrete-time, discrete-state representation of the problem. In the simulation, each object performs high-level actions, such as moving forward from one location on the road to the next allowable location forward, travelling through an intersection, making a request for information, sending a message from one node to another, etc. In this way we can define a fundamental time step (e.g. 40 ms), and we can control the speed at which actions are performed by specifying some integer multiple of time steps that must pass before performing another action. In our simulations for example, sending a message between sensor nodes only takes one time step, whereas cars need to wait 25 time steps before moving forward to the next position along the road. In this way, we are able to let the pursuers move 67% faster than cars by setting their update time to every 15 time steps. Note that some actions are allowed to be performed simultaneously, such as a pursuer making a request for information and moving through an intersection.

Originally we implemented this update strategy by having each object decide when it needs to update. To do this, each object stores its own simulation time, and at each time step, every object would compare its time with the next time that it gets to act, and if that time has come, it performs its action. In our setup, there were 192 sensor nodes that would spend more than 90% of their time being idle, yet they had to check at every time step to see if they needed to be updated, which incurred a significant computation overhead. To improve this situation we implemented a Java PriorityQueue through the interface MATLAB has to Java. In this setup, at each time step the elements whose time it is to act are removed from the front of the queue and updated. After they perform their actions, they determine their next update time, and add themselves back into the PriorityQueue, sorted by this update time. This update improved the speed of the simulation by a factor of three.

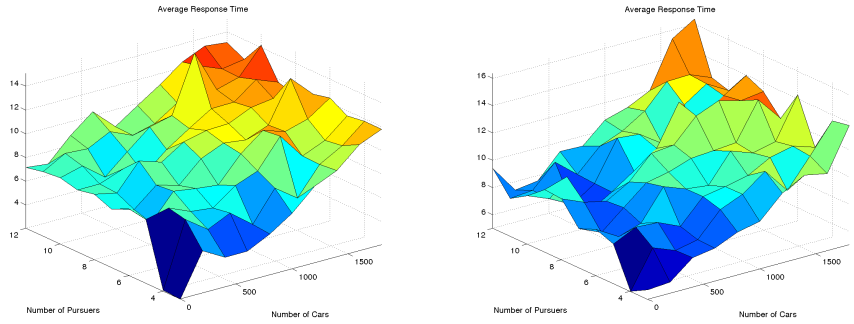
8 Simulation Specifications

The city grid in our simulation is 15 blocks long and 11 blocks wide wherein each block is 50 meters in length. The road has been configured as having 2 lanes with one going forward and the other going backwards. The other parameters that we vary through various test scenarios are number of pursuers and number of normal cars to account for various congestion scenarios. The maximum speed limit has been set to 50 kph. The pursuer and the evader car both moves at the same speed which is approximately 5/3 times the speed of the normal cars.

9 Results

Various performance parameters were compared for the simple as well as smart Pursuer algorithm. The average response time of sensor was found to increase in both the cases with increase in number of pursuer and normal cars which

can be explained as due to slowing down of evader due to more congestion and also because more pursuers are asking the network for information, as seen in figures 6a, 6b.



(a) (b)

Figure 6: (a) Algorithm 1: Average Response Time vs Number Of pursuers and Normal Cars (b) Algorithm 2: Average Response Time vs Number Of pursuers and Normal Cars

The idle time percentage of sensors decreased with increase in the number of pursuer cars. This is seen in figures 7, 8.

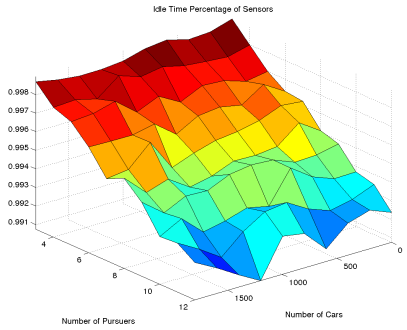


Figure 7: Algorithm 1: Sensor Idle Time vs Number Of pursuers and Normal Cars

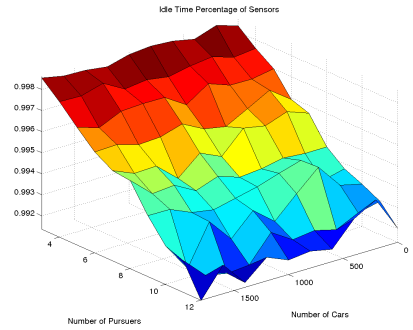


Figure 8: Algorithm 1: Sensor Idle Time vs Number Of pursuers and Normal Cars

The maximum number of messages sent by the sensor and number of messages per time slot also increased in both the cases with increase in number of cars on the road as seen in figures 9,10. The minimum idle time of sensor was

actually slightly more in case of the smart algorithm as compared to the simple one.

Figure 9: Algorithm 1: Max Messages Sent by a Sensor vs Number Of pursuers and Normal Cars

Figure 10: Algorithm 2: Max Messages Sent by a Sensor vs Number Of pursuers and Normal Cars

100 percent convergence was achieved for all the test cases in case of the smart algorithm; however the simple algorithm failed to converge on few occasions with a certain configuration, as seen in figures 11,12.

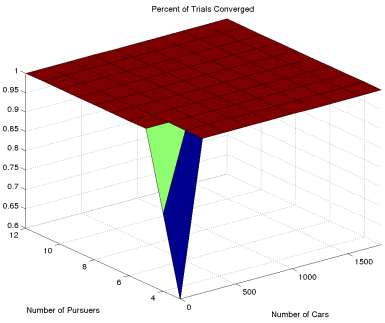


Figure 11: Algorithm 1: Number of Trials Converged vs Number Of pursuers and Normal Cars

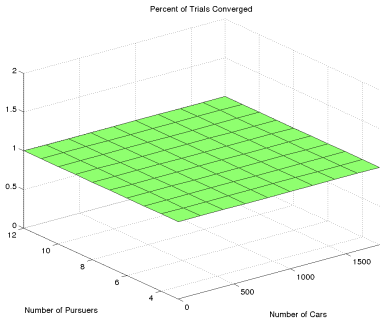


Figure 12: Algorithm 2: Number of Trials Converged vs Number Of pursuers and Normal Cars

As expected, The number of time steps to converge was lower in case of the smart algorithm and so was the total number of messages. This is seen in figures 13,14.

Figure 13: Algorithm 1: Time Steps to Convergence vs Number Of pursuers and Normal Cars

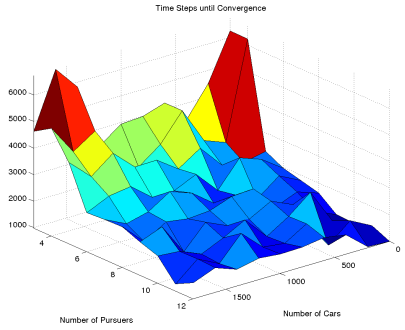


Figure 14: Algorithm 2: Time Steps to Convergence vs Number Of pursuers and Normal Cars

It was also observed that as the number of cars increased on the road, the worst response time was higher for the simple algorithm as compared to the smarter one, as seen in figures 15,refWorst Response Time 2.

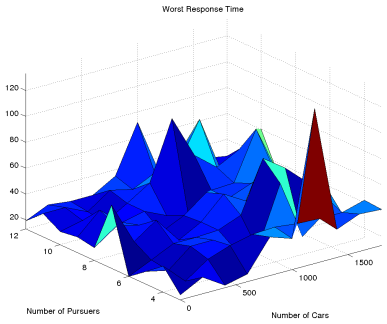


Figure 15: Algorithm 1: Worst Response Time vs Number Of pursuers and Normal Cars

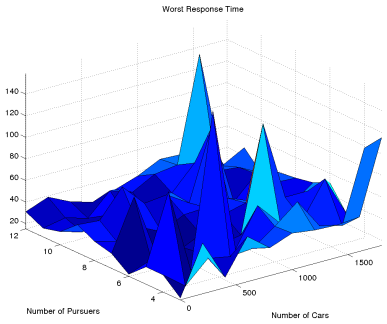


Figure 16: Algorithm 2: Worst Response Time vs Number Of pursuers and Normal Cars

We also increased the evader speed compared to the pursuer speed and observed the number of time steps to convergence and percentage of converged trials as a whole. The smart algorithm clearly performed better than the simple one This is seen in figures 17,18.

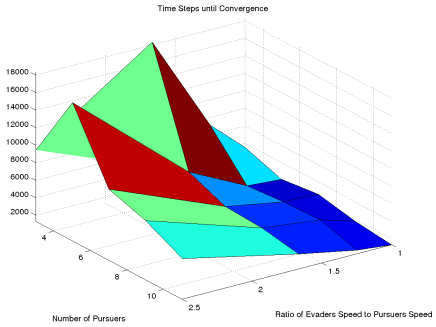


Figure 17: Algorithm 1: Time Steps to Converge vs Number Of pursuers and Ratio of Evader to Pursuer Speed

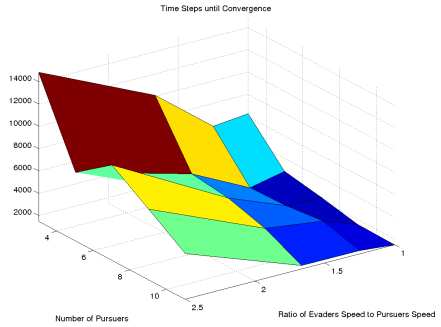


Figure 18: Algorithm 2: Time Steps to Converge vs Number Of pursuers and Ratio of Evader to Pursuer Speed

10 Conclusion

In this project, the problem of developing algorithms that incorporate stationary sensor information, communication systems and on-board path planning has been successfully tackled. Two different pursuit algorithms have been tested, name 'simple path planning' and 'smart path planning'. It has been demonstrated that the average time taken to intercept the evader is lower for the smart algorithm and it also increases the interception efficiency. The limited time frame of the project made it difficult to test a lot of the other algorithms that had been designed.

Given that the problem is first tackled in this project, a large number of possible directions of research have been identified. In order to make the problem more realistic, the individual pursuer vehicles could be allowed to communicate with each other as in the case of a real pursuit. This is likely to enable a higher degree of co-ordination between the pursuer vehicles and will in all likelihood decrease the interception time. It has also been observed that in order to enable advanced path planning, evader trajectory estimation and congestion estimation would be required. Congestion estimation is an existing field of research with a large body of work been done on it already. Moreover, adding resource constraints for the pursuer's and the evader (like fuel constraints) would make the problem more realistic.

Another direction of research is exploring the effect of having a leader who co-ordinates the path planning of the other pursuers [2]. This too would require a significant amount of estimation of congestion conditions and the evaders trajectory in order to best optimize the the path planning. The effect of dynamically changing leaders on the interception time would also be an interesting study.

Finally, identifying the best possible pursuit algorithm using machine learning can be done to see if the pursuit algorithms present taken by police officers in such pursuit scenario's is indeed the most efficient one.

References

[1] A. Isaza, J. Lu, V. Bulitko, and R. Greiner. A cover-based approach to multi-agent moving target pursuit. *Artificial Intelligence and Interactive Entertainment COnference, AIIDE*, oct 2008.

[2] N. Leonard, T. Shen, B. Nabet, L. Scardovi, I. Couzin, and S. Levin. Decision versus compromise for animal groups in motion. *Proceedings of the National Academy of Science of the United Stated of America*, 109(1):227–232, 2011.

[3] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *Personal Communications, IEEE*, 7(5):16 –27, oct 2000.

[4] Cagatay Undeger and Faruk Polat. Multi-agent real-time pursuit. *Autonomous Agents and Multi-Agent Systems*, 21(1):69–107, July 2010.

[5] Ermin Wei, Eric W. Justh, and P.S. Krishnaprasad. Pursuit and an evolutionary game. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 465(2105):1539–1559, 2009.

[6] D. Zu, J.D. Han, and M. Campbell. Artificial potential guided evolutionary path plan for multi-vehicle multi-target pursuit. In *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, pages 855 –861, aug. 2004.

11 Additional Figures

Figure 19: Algorithm 1: Max Messages per Time Slot vs Number Of pursuers and Normal Cars

Figure 20: Algorithm 2: Max Messages per Time Slot vs Number Of pursuers and Normal Cars

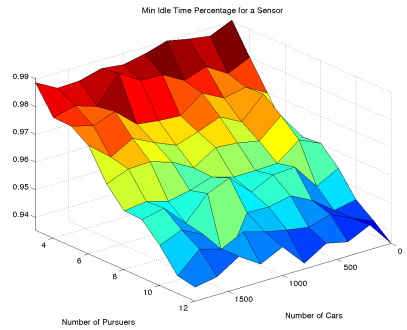


Figure 21: Algorithm 1: Min Idle Time vs Number Of pursuers and Normal Cars

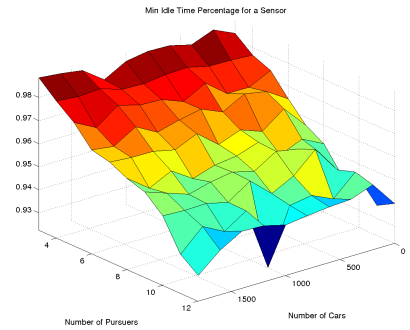


Figure 22: Algorithm 2: Min Idle Time vs Number Of pursuers and Normal Cars

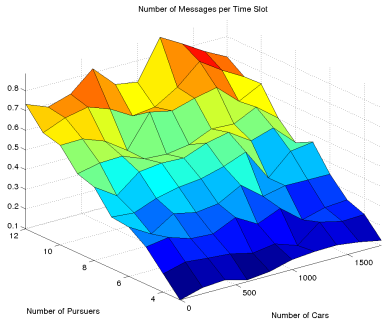


Figure 23: Algorithm 1: Number of Messages vs Number Of pursuers and Normal Cars

Figure 24: Algorithm 2: Number of Messages vs Number Of pursuers and Normal Cars

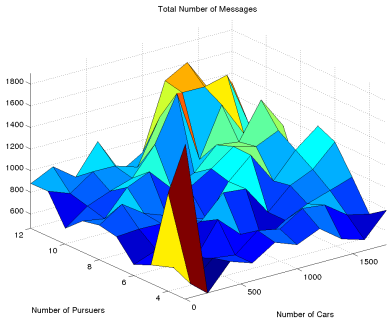


Figure 25: Algorithm 1: Total Number of Messages vs Number Of pursuers and Normal Cars

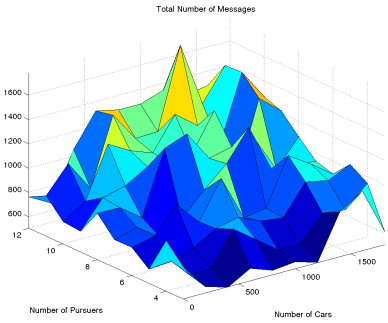


Figure 26: Algorithm 2: Total Number of Messages vs Number Of pursuers and Normal Cars

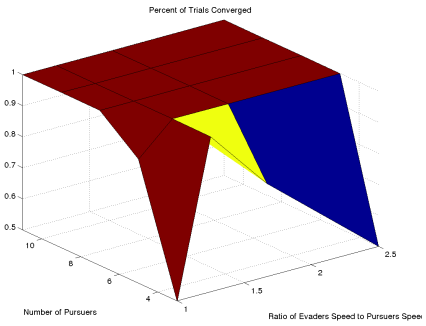


Figure 27: Algorithm 1: Number of Converged Trials vs Number Of pursuers and Ratio of Evader to Pursuer Speed

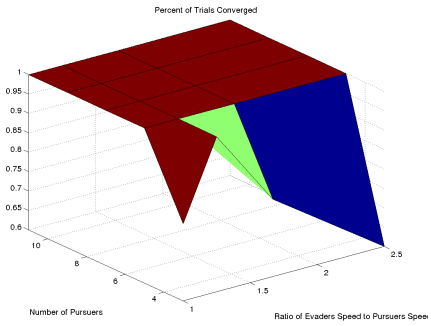


Figure 28: Algorithm 2: Number of Converge-ed Trials vs Number Of pursuers and Ratio of Evader to Pursuer Speed