Vehicle Trafiic and Formation

GROUP 4

Leran Li Peiheng Li

20010324 20010315

leran@kth.se peiheng@kth.se

Abstract

In multi-agent systems, efficient traffic navigation and formation maintenance are crucial. For traffic, the focus is on dynamically avoiding other agents while moving towards the target. Previously, Reciprocal Velocity Obstacle (RVO) has been proposed for real-time multi-agent navigation, typically focusing on pair-wise collision avoidance. Optimal Reciprocal Collision Avoidance (ORCA) has been introduced to provide efficient collision avoidance between multiple pairs of agents. For formation, the emphasis is on the collective movement of agents. Behavior-based Formation Control is a method for maintaining formation while still allowing agents to avoid obstacles and other agents. In this work, we present an implementation of these methods. Specifically, we integrate the methods with vehicle control under their kinematic constraints and extend their application to scenarios involving more agents. Finally, we demonstrate that our solutions yield reasonably good results under different environments.

1 Introduction

The challenges this work is trying to solve are about controlling the behaviours of multi-agents in the same environments. In more detail, we need to implement two separate systems to solve the traffic problem and formation problem separately. For traffic, we have a set of agents with their corresponding starting points and goals. Each agent needs to move from the starting point to the goal while avoiding colliding with others. For formation, we have a group of agents and a set of checkpoints. The group of agents is required to pass each checkpoint with a minimum time error and finish all the checkpoints within the minimum time. By solving those two problems, one can use the solutions to move a collection of agents in the same environment for collision avoidance and cluster formation purposes, which is the key to a multi-agent system.

The focus of our work is to tackle dynamic collision avoidance and formation-keeping. The methodologies for motion planning and individual car control are adapted from prior work in vehicle racing. For dynamic collision avoidance, we employed the Optimal Reciprocal Collision Avoidance. For formation keeping, we implement the behaviour-based formation control.

Among the 32 groups, in terms of the total time taken for both problems, our solution achieved the 7th rank. This performance indicates that our approach yields a reasonably effective solution to the posed problems.

1.1 Contribution

In this work, we present a stable and fast solution for traffic problems by collaborating A* with ORCA. We proposed modifications that make the ORCA more suitable for our agent behaviors, resulting in great results for drones and fairly good results on cars. We also present a novel way for the formation problem based on behavior-based formation control. Generally, our methods proved to be robust and effective.

1.2 Outline

In Section 2, we review the related study in traffic safety and formation-keeping. In Section 3, we explain the details behind our implementation of Optimal Reciprocal Collision Avoidance for the traffic problem and the behaviour-based formation controller for the formation problem. In Section 4, we present the evaluation results and discuss the factors that contribute to the performance. We also compare and analyse the solutions from

other colleagues to discuss potential improvements. Finally, in Section 5, we conclude this work by summarising our contribution.

2 Related work

In this section, we give a brief overview of related work on navigating agents in dynamic environments for traffic safety and formation-keeping purposes.

For traffic, numerous solutions have been developed for agents to solve the traffic problem, aiming to complete the target position without colliding with each other. One of the main ideas is to find a globally safe path, treating other agents as dynamic obstacles [1]. Such a method can be computationally heavy but easy to carry out. An alternative to completing the planned path is to plan for the agent as it acts, taking real-time feedback from the surroundings and making local decisions as they arrive. Most of the effective solutions are based on the idea of *Velocity Obstacles* (VO) [2]. The general idea is to let the agent make decisions based on the information in the velocity space. More specifically, VO can help the agent define a set of velocities that would lead the agent to collide with other moving obstacles in the future. In multiple-agent collision avoidance situations, original VO has many problems, such as oscillation, non-optimal velocity choice, and so on. Other solutions have been proposed to solve these problems. The *Reciprocal Velocity Obstacle* (RVO) [3] helps solve the oscillation problem. It is also the foundation of further ideas. RVO lets each agent take half of the decision weight, thus showing promising results even with more than two agents. The *Optimal Reciprocal Collision Avoidance* (ORCA) [4] takes account of every other moving obstacle's VO, picking out the best possible velocity that is closest to the desired velocity through linear programming. There are also more ideas about more specified situations. The *Acceleration Velocity Obstacle* (AVO) [5] considers the acceleration in one's velocity space. While it is theoretically solid, the implementation is computationally heavy and limited to hardware constraints.

Several previous studies have addressed this problem under different assumptions about the environment. [6] introduce the virtual structure control scheme, which provides a stable and robust result. However, this method cannot be adopted directly in an environment with obstacles. The formation-leader obstacle (FLO) [7] proposes a control scheme to avoid obstacles when keeping the formation. By introducing the FLO set, this scheme considers the obstacles to all the agents when planning the path for the leader. Given that, a controller is precise enough and the path exists, such a scheme promises formation-keeping and obstacle avoidance. However, this method is limited

to static formation and does not guarantee finding a solution if the FLO set is fully blocked. [8] introduces a more general approach, the behaviour-based control scheme. Upon different scenarios during the path execution, agents have different behaviours, including moving to the goal, keeping formation, and avoiding others.

3 Proposed method

In this section, we present our solution to the two problems respectively. Then, we highlight some implementation decisions.

For the traffic problem, our goal is to let every agent reach the target position as soon as possible. Our method uses A* for global path planning (from Group 19, A1) and ORCA for local collision avoidance. The final algorithm integrates these two main sectors with small modifications to suit map situations and special agent behaviours.

For the formation problem, our approach using the RRT* (from Group 21, A1) for path planning, integrates it with the behaviour-based control scheme. The solution consists of three parts, formation keeping, moving to the goal and avoiding others.

3.1 Traffic

To reach our goal for the traffic problem the first part is to delve into the collision avoidance method. We want the agents to avoid colliding with each other as much as possible so that they can maintain the ability to move. The second part is integrating the collision avoidance method with our pathfinding method, letting the agent move toward the target heuristically.

3.1.1 Delve Into Collision Avoidance

One of the essence of the traffic problem is collision avoidance. We focused on the study of using Velocity Obstacles (VO) for real-time collision avoidance, that is, treating other agents as moving obstacles, sensing the mutual velocities as well as positions, making decisions in the subset of the space that won't form collision in the desired future. More specifically, our method follows the ORCA discussed in [4], with modifications that suit our environment and agent behaviors. We first give the definition and illustration of the velocity obstacles, then describe how an agent calculates new velocities.

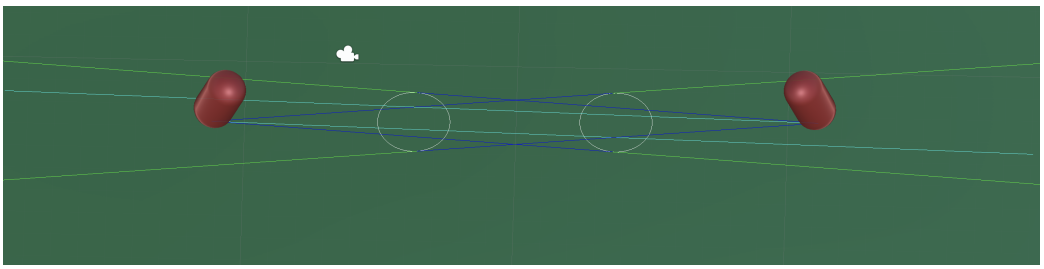

Figure 1: VOs of a pair of drones approaching each other

Velocity Obstacle The velocity obstacles in our method can be defined as:

$$VO^{\tau}_{A|B} = \{v|\exists t \in [0,\tau] :: tv \in D(p_B - p_A, r_A + r_B)\}$$

And in which D is a set of points in such a disc defined as:

$$D(p,r) = \{q|||q-p|| < r\}$$

The definition above means the velocity obstacle for A induced by B for time window τ is a set of points (or vectors, depending on the environment) that fill in the constraints.

An illustration in Unity for the velocity obstacle of a pair of drones approaching each other is shown in figure 1. The VO is calculated in velocity space, thus the coordinates shown in the physical space don't represent anything. We can simply show VO originates at each agent's central position. In figure 1, the cyan straight lines are relative velocities, and the white circles are the τ cut-off discs $D(\frac{p_B - p_A}{\tau}, \frac{r_A + r_B}{\tau})$. The light green rays are the left and right legs of the VOs, as in the definition above. They can be interpreted as the continuous set of the tangent points of cut-off circles at different time steps. The blue lines just help to identify the VO belongings.

Optimal Reciprocal Velocity Calculation We need to calculate the velocity that can let the agents avoid collision with each other based on VOs. Based on the preferred velocity, our collision avoidance method will assign the agent a new safe velocity, after considering all the other agents' velocity obstacles.

Our method for calculating the optimal velocity is visualized in figure 2. We describe the general idea of this method taking figure 2 left as reference. Once we have $VO^{\tau}_{A|B}$ and the relative velocity, we can find the shortest vector that leads the relative velocity to be out of the $VO^{\tau}_{A|B}$. This is the vector **u**. The "reciprocal" stands for letting agent A and agent B each take half

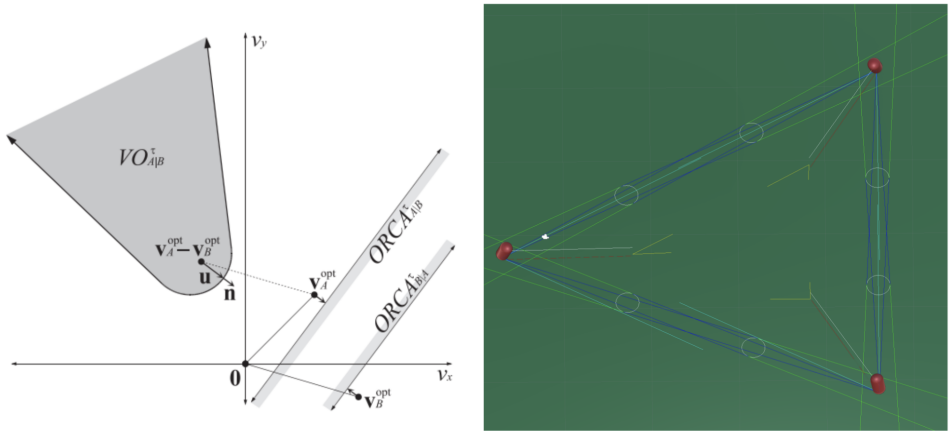

Figure 2: Optimal Velocity Calculation. *Left:* ORCA described in [4]; *Right:* Optimal change for each introduced VO, short yellow lines in the middle.

of the $\mathbf{u}$. Assume we only have agents A and B, they are both following the preferred velocity ($\mathbf{v}_A^{opt}$ and $\mathbf{v}_B^{opt}$) approaching each other. Then adding half of $\mathbf{u}$ to each of the preferred velocities will make sure they don't collide with each other in τ. However, we are dealing with more than two agents, which means we will have a lot of $\mathbf{u}$ to deal with. The solution for this problem is that instead of storing the $\mathbf{u}$, we tend to store the half-plane that divided the whole velocity space by the vector that is perpendicular to $\mathbf{u}$. It is easy to prove that every vector selected in the half-plane that $\mathbf{u}$ points to will assure the collision-free criteria. This is the formation of the $ORCA^{\tau}_{A|B}$ line shown in figure 2 left. And $ORCA^{\tau}_{B|A}$ line is defined symmetrically. More formally, we can have the ORCA defined as:

$$ORCA^{\tau}_{A|B} = \{\mathbf{v} | (\mathbf{v} - (\mathbf{v}_A^{opt} + \frac{1}{2}\mathbf{u})) \cdot \mathbf{n} \geq 0\}$$

$$ORCA^{\tau}_{B|A} = \{\mathbf{v} | (\mathbf{v} - (\mathbf{v}_B^{opt} + \frac{1}{2}\mathbf{u})) \cdot \mathbf{n} \geq 0\}$$

With multiple agents, one can stack all the half-planes together, and find the intersection of all. This is done by linear programming [4], which is essentially an $O(n)$ solution.

3.1.2 All About Collaboration

The collaboration algorithm is given in algorithm 1. We used the A* algorithm for the agents to find the path towards the targets in complex environments. Then the position error is used to calculate the preferred velocity,

Algorithm 1: A*-ORCA Collaboration

```
Define thisAgent, status;
Define positionError, preferredVelocity, safeVelocity, constraints, velocityError;
foreach fixedUpdate do
    otherAgents ← getOtherAgents();
    status ← getCurrentStatus(thisAgent);
    lookAheadResult ← proportionalLookAhead();
    foreach otherAgent in otherAgents do
        if withinRange(thisAgent, otherAgent) then
            speedScale ← scaleMaxSpeed(thisAgent, otherAgent);
            if inQueue(thisAgent, otherAgent) then
                Brake(thisAgent) ;                    // Stop the agent
            end
        end
    end
    preferredVelocity ← getPreferredVelocity(positionError);
    safeVelocity ←ORCA(status, preferredVelocity, constraints);
    PD_Control(velocityError, positionError);
end
```

which is the velocity one should take when there are no other moving obstacles. We then use the current agent's properties such as current velocity as well as current position, the preferred velocity, and other agent's information to calculate a collision-avoiding velocity through ORCA. This velocity with position error determined the desired acceleration for both cars and drones through the PD controller.

The A* path planning is done at the beginning of the game, which behaves the same as the one used in Assignment 1, Group 19's implementation.

After making sure the A* and ORCA are both functioning, we need to coordinate them. Simply connecting the primitive PD controller with ORCA can lead to unsafe navigation and slow movements. Some general differences between the algorithm 1 and the primitive PD controller are:

- Proportional look ahead. New safe velocity can lead one very biased from the desired path. Using the proportional look ahead method for tracking the path is the stable solution.
- Max speed scale. When it comes to crowdy, an agent usually tends to slow down for more stable behavior.
- In-queue stopping. Typically for a very narrow path, it is not desired to avoid each other too much, since it usually comes to hitting the wall. Introducing a stopping criteria solves this problem, letting one agent pass at a time.

- ORCA valid distance. This is a modification that is typically suitable for speed-up. When other agents are far away, it is not necessary to calculate the velocity obstacle in between.

3.1.3 Other Attempts

Except for implementing ORCA for moving obstacles, we have also tried to use some other methods. They are not functioning well, but worth mentioning.

Static ORCA Static ORCA easily treats the static obstacles as the moving obstacles with a velocity of zero and sets the τ to be low. It works in very open areas. However, in many situations the static obstacle's VO tends to block the path, leading to a contradiction of finishing the path.

Acceleration Velocity Obstacle In velocity space, if we consider the acceleration of an agent we can have a more complicated form of VO definition. In theory, AVO gives a better prediction of the agent's behavior since we are controlling the agent's acceleration. However, there's no closed-form solution for quickly calculating the AVO of each pair of agents. Thus it is a super computationally heavy algorithm, almost unreachable in a simulation environment such as Unity. Furthermore, this method might lead to oscillations since acceleration can be applied from both left and right, which is not stable.

3.1.4 Implementation

Our implementation for the traffic problem mainly consists of two parts as well, the A* path planning section as well as the ORCA. As mentioned above, A* path planning extends the implementation from Group 19, Assignment 1, which is a grid-based searching algorithm. It shows great robustness, there wasn't much modification. The ORCA, with reference to [4], was implemented from scratch. Partial linear programming implementation took the reference of the implementation of the [4] originally, but since it is not solving the general problem, it doesn't conflict with the interest of the topic.

3.2 Formation

The overview of our approach is described as follows. Each agent plans their path between each pair of checkpoints and combinates into a path from the starting position to the last checkpoint. Then execute the path using

Algorithm 2: Behaviour Based Controller

```
1  foreach update do
2      if CloseToGate(i, m_gate) then
3          if AllReady() then
4              i ← i + 1;
5              thisAgentReady ← False;
6          end
7          else
8              if |v| = 0 then
9                  thisAgentReady ← True;
10             end
11             else
12                 Brake();
13             end
14         end
15         return;
16     end
17     if CarNearBy(m_agent) then
18         Brake();
19         return;
20     end
21     FollowThePath();
22 end
```

the behaviour-based controller. Algorithm 2 describes the details of this controller. The various functions are defined as follows:

- `CloseToGate(`i, m_{gate}`)`: Check if this agent is close to the i-th group of the gates with the distance of m_{gate} units.
- `CarNearBy(`m_{agent}`)`: Check if there is any other agent close to this agent within m_{agent} units.
- `AllReady()`: Check if all agents has boolean flag $thisAgentReady = True$.
- `Brake()`: Apply brake.
- `FollowThePath()`: Execute the path directly.

The main difference between our approach and the origin one described in [8] is that we use if-else to control the different behaviours explicitly and only one behaviour is executed in a given time. The reason we made such a choice is that formation only matters at the checkpoints in our problem's scope. Explicitly separating the consideration of formation and moving to the next checkpoint is more suitable for the aim described in Section 1.

3.2.1 Implementation

There is a trick we employed in the implementation of formation. We extend the RRT* and the controller from Group 21 in A1. This motion planner includes a method to bake the velocity in the path. We modified the code such that when planning each segment of the path between each pair of checkpoints, the velocity in front of some distance of the checkpoints is 0. This guarantees all the agents will stop in front of the gate smoothly.

4 Experimental results

In this section, we first describe the setup of the experiment, including the test cases and the metric of evaluation. Then, we present the difference in performance between our solution and the best solutions. Finally, we analyze and discuss such differences, separately for traffic and formation problems.

4.1 Experimental setup

Both traffic and formation problem shares the following common setup. The experiment consists of different environments (terrains) to test on, and the metric is the time taken to finish the terrain. The experiments were conducted using two agents, each operating under one of two kinematic models: car and drone. The traffic experiments for cars and drones are denoted as P1 and P2, respectively, while the formation experiments are referred to as P3 and P4.

For traffic problems, it consists of five different terrains. The time taken to finish one terrain is measured by the time elapsed from the beginning until the last agent reaches its corresponding goal. In the formation problems, two test terrains were used. These terrains varied in checkpoint configurations. One of the terrains has obstacles. The performance metric here combines the total penalty time, calculated as the sum of time differences between agents passing checkpoints, with the overall time taken to clear all checkpoints.

4.2 Analysis of Outcome

The experiment result indicates that our solution yields a reasonably good result with an overall time taken rank of 7th. The comparison of our solutions to the best solutions is shown in Table 1 - 4. One can observe that our solution of P2 is close to the best solution with a small gap, while the result of the rest of the solutions implies a space for improvement.

	open	semi_open	intersection	highway	onramp
Ours	61.1	73.72	79.56	80.4399	79.52
Best	18.98	22.6	50.1	65.62	43.34

Table 1: Comparison between our solution on P1 Final to the best solution

	open	semi_open	intersection	highway	onramp
Ours	20.72	18.9	30	41.68	41.96
Best	17.62	18.74	29.22	41.02	41.82

Table 2: Comparison between our solution on P2 Final to the best solution

	6	6B
Ours	174.6	89.62
Best	90.28	67.53

Table 3: Comparison between our solution on P3 Final to the best solution

	6	6B
Ours	149.62	78.34
Best	84.96	64.2

Table 4: Comparison between our solution on P4 Final to the best solution

4.2.1 Traffic

We have a generally satisfying result for the traffic problem. Although P1 has space to further speed up, we have almost the best P2 performance, with every terrain approaching the best result within 0.8s on average (it is worth noting that the best results are from different groups).

For the P1, we have to admit that ORCA has many constraints on car-like agents. Comparing the open and semi-open results with Group 3, we can easily see that the time consumption difference is huge. Group 3 planned the globally collision-free path, which was very hard to do but very fast to execute. Our ORCA-based method tends to give a stuck in the middle and solve out one by one slowly. The intersection, highway, and onramp share some common characteristics. Compared with the best of them, we lack some other methods, such as replanning for less possible collision and a better highway path merging mechanism.

For the P2, although we are close to the best result, there are still some reasons that make us 1s slower. Typically compared with Group 26, we

realized that for holonomic agents such as drones, a simple VO selection solution can give a shorter path than ORCA. ORCA is more about being completely collision-free, thus may lead to a longer path to go. However, one can speed up if more "suitable" collisions happen.

4.2.2 Formation

To summarise the formation performance, based on results from Tables 3 and 4, shows that our solution closely approaches the best performance in the terrain with obstacles and fewer checkpoints (6B_final). However, it falls significantly behind in the terrain with more checkpoints (6_final), for up to about twice the time taken to the best.

Three main factors contribute significantly to the result. For the time difference of passing the checkpoints, it implies the amount of penalty time received. Our solution forces the agents to stop in front of the checkpoints and then move together so that only a small amount of penalty time is received. In terms of avoiding collision with others, the existence of collision means agents will waste time on returning to the path. Our solution will slow down the agent to guarantee avoidance. Moving to the goal affects the total time taken to finish all the checkpoints. This is the shortcoming of our solution because we choose to stop in front of the checkpoints, which trades the penalty time with the longer time taken to finish.

In comparison to the best solutions provided by groups 2, 6, 7, and 11, all of those solutions do not stop the agents but dynamically plan the speed while executing the path. Though their solution likely receives a higher penalty time, the overall time taken is lower. In more detail, those solutions either use the virtual structure to plan the agent's speed implicitly or calculate the desired speed explicitly. Such observation suggests that one potential improvement is to integrate virtual structure into our solution.

5 Summary and Conclusions

In summary, we implemented A* with ORCA for the traffic problem and developed a behavior-based formation control system for the formation problem. Based on the work from the last assignment, our method completed all the tasks. For the traffic problem, ORCA and A* showed great results on the drones, with quite promising results with car agents with our modification that made the algorithm more versatile. For the formation problem, we have shown great potential with our behavior-based formation control system, yet with space to further speed up.

References

[1] K. Kant and S. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *International Journal of Robotic Research - IJRR*, vol. 5, pp. 72–89, 09 1986.

[2] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, pp. 760–, 07 1998.

[3] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1928–1935.

[4] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19.

[5] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3475–3482.

[6] B. Young, R. Beard, and J. Kelsey, "A control scheme for improving multi-vehicle formation maneuvers," Feb. 2001, pp. 704–709 vol.2.

[7] P. Ogren and N. Leonard, "Obstacle avoidance in formation," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Taipei, Taiwan: IEEE, 2003, pp. 2492–2497. [Online]. Available: http://ieeexplore.ieee.org/document/1241967/

[8] T. Balch and R. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, Dec. 1998. [Online]. Available: http://ieeexplore.ieee.org/document/736776/