Assignment 3 Report
DD2438

GROUP 3:10

Nino Segala
981208
segala@kth.se

Fredrik Diffner
880629
fdiffner@kth.se

Abstract

How can several agents collaborate to achieve the best performance? In this work, we present approaches to two common but still hard tasks; the task of collision avoidance between multiple autonomous agents and the task of playing a soccer-like game against another team of autonomous agents. This is done for drone-like agents which can travel in the horizontal plane. For the first problem we use Velocity Obstacles for local planning, the A* algorithm for global path planning, and a PD-controller for path following. For the soccer problem, we present a Behaviour Tree inspired by the tactics used in the game Rocket League. Both solutions perform well, among the top 5 when comparing to other groups in the course. The solutions are also flexible and could easily be extended to other problem settings.

1 Introduction

The first problem, collision avoidance between multiple autonomous agents, is an important problem relevant for several fields such as autonomous driving, robotics, as well as computer AI and simulations. There are several different approaches, which some may be more or less well suited to specific problem spaces. The problem setting consists of 50 agents starting in a big circle, and simultaneously want to travel to a certain goal position located at the opposite side of the circle. The space the agents travel in could either be free from obstacles or have obstacles forming a narrow four-way corridor. The problem is considered solved when each agent has reached their goal position without too many collisions. Our solution is based on the use of Velocity Obstacles [2] in combination with a PD controller [7], and the A* algorithm [3] for global path planning when needed. With this approach, we receive the second-best time on the open-field map among the groups in the course, as well as the 5th best time on the map with obstacles.

The second problem, multi-agent drone soccer, highlights the increasing game AI industry. AI is used in video games since the 1950s and has a massive role in the actual games, like in the soccer game FIFA for example where the players who are not controlled by the human playing the game need to be controlled by an AI. It is also often possible to play against the AI, so the opponent team or enemies have to be handle by an AI. AI has also a significant impact on sports for strategic decisions for example. Nowadays AIs are analyzing data collected from the games, but we can easily imagine how simulations could lead to finding new winning strategies. The soccer game consists here of two teams of three drones. The game lasts 100 seconds, at the end the team which has scored the most goals wins and in the case where the score is equal, it is reported as a draw. Our solution consists of implementing a behaviour tree (BT), introduced by Dromey in 2001 in [5], controlling one agent and which leads to a global strategy. We used a PD controller [7] for most of the moves. We finished second out of thirteen teams in the final soccer game.

1.1 Contribution

In our work, we present a generic approach for collision avoidance using a simplified implementation of Velocity Obstacles in combination with a PD controller. We also present a well-performing behaviour tree for a soccer-like game with autonomous agents, showing that a fixed goalie is not a requirement to perform well.

1.2 Outline

In Section 2 we examine some of the related work for the two problems, and in Section 3 we go more into detail of our implementations. In Section 4 we present and analyze the results of our implementations, and compare them with the results from other groups in the course. The work is summarized in section 5, where also suggestions for improvements are discussed.

2 Related work

In this section related work to the two problems are presented. First previous research related to the problem of collision avoidance is examined. After that, researches related to the soccer problem are presented.

Several proposed methods in the field of collision avoidance between multiple agents are based on the theory of Velocity Obstacles, presented in [2]. The authors of [9] builds on the work in [2]. They show that the concept of Velocity Obstacles leads to oscillations in multi-agent navigation problems, where the agents independently take actions to avoid collisions with other agents. To address this problem, they present the concept of Reciprocal Velocity Objects where robots are assumed to follow the same navigation rules. Hence a robot about to collide with another robot can assume that the other robot will take partial responsibility for avoiding the collision. According to [6] this could still lead to problems, in this case, called "reciprocal dances", which under extreme circumstances may cause the algorithm to never converge. The authors of [6] introduce Hybrid Reciprocal Velocity Obstacles, which encourage the robots to pass on the correct side of each other. Another direction for the problem of collision avoidance between multiple agents is to model the agents as a flock. Craig W. Reynolds explores the behaviour of single agents acting in flocks in [4]. He presents 3 simple behaviours which each actor can follow to form a flock motion: Collision Avoidance, Velocity Matching, and Flock Centering.

In [8] the authors propose a method for an Unmanned Aerial Vehicle following a moving target on the ground. The method is inspired by switching between the two methods Proportional Navigation (PN) and Proportional Derivative (PD). They claim that their solution is robust and effective compare to only using a PD controller. The idea of Behaviour trees was first introduced by Dromey in 2001 in [5]. It is a good tool for artificial intelligence in computer games replacing slowly the finite state machines introduced by

E. F. Moore. The behaviour trees are presenting several advantages explaining their recent popularity. Those can be found in the book [1] written by M. Colledanchise and P. Ögren in 2020 and the main ones are the modularity, the readability and that it can be easily analysed. BTs still present some disadvantages like the complexity it can have in some case and the fact that it is a recent model which still needs development.

3 Proposed method

This section will go into more detail about our approach. For Problem 1 we will present a generic approach not dependent on specific start or goal positions nor on a specific structure of the map agents travel in. Further down we present the solution for Problem 2, inspired by tactics from the E-sport game Rocket League [1].

3.1 Problem 1: Collision avoidance

First, the path for each agent is planned without other agents taken into account, which mimics a real-world scenario when an agent doesn't know the exact plan of each other agent. If the agent moves in a space without static obstacles, this path only consists of the start and goal position. If there is a space with static obstacles, the path is planned with A* [3] and then displaced a bit to the right relative to the forward travel direction for each agent to simulate right-hand traffic. Two "speed limits" are also implemented; one more liberal for open spaces without many other agents close by, and one lower when there is a lot of other agents in the neighbourhood. Each agent then uses a PD-controller [7] to follow the assigned path. Since the agents start in a circle shape and simultaneously wants to move to the opposite side of the circle, several collisions will occur in the middle without a collision avoidance method.

3.1.1 Velocity Obstacles

Since the agents have access to other agents position and velocity, Velocity Obstacles [2] is a suitable method for this problem setting. Briefly, a velocity vector pointing into a Velocity Obstacle (VO) will lead to a collision at some point. The goal is to choose velocities that lay outside of VO's.
We let a disc-shaped agent A have radius r_A, position $\mathbf{p}_A$, goal $\mathbf{p}_A^{goal}$, current

[1] https://www.rocketleague.com/

velocity $\mathbf{v}_A$, and preferred velocity $\mathbf{v}_A^{pref}$. The velocity obstacle $VO_{A|B}$ induced by agent B for agent A is the set of all velocities of A that will lead to a collision at some moment in time with agent B assuming that B maintains its current velocity. Formally:

$$VO_{A|B} = \{\mathbf{v}|\exists t > 0 : t(\mathbf{v} - \mathbf{v}_B) \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\}$$

where $D(\mathbf{p}, r)$ represents a disc with center $\mathbf{p}$ and radius r. $VO_{A|B}$ will form a cone with its apex at $p_A + v_B$, which can be seen in figure 1.

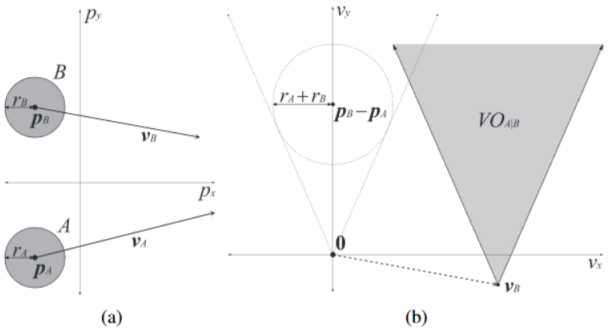

Figure 1: The two agents A and B with position $\mathbf{p}$, radius r and velocity $\mathbf{v}$ in a) create the velocity obstacle $VO_{A|B}$ shown in b). Picture from [6].

If a preferred velocity is located inside a VO, we need to find a new velocity to avoid collisions, and there are several ways to do this. In [6] they claim that the optimal new velocity (optimal in such a way that it is closest to the preferred velocity) could be found in a set of candidates. This set consists of intersections between VO's which do not lay inside a VO, and the projections of the preferred velocity on VO's which do not lay inside a VO. If the travel space also contains obstacles, we also need to ensure that the candidates do not lay inside or too close to an obstacle. The candidate which is closest to the preferred velocity is then the new optimal velocity. This is illustrated in figure 2. Since this new velocity may cause the agent to lose sight of the next waypoint in the path due to static obstacles, we may need to recalculate the path with the A* algorithm.

The approach of Velocity Obstacles also allows for incorporating kinematic constraints on the agents, although this is not considered here due to time constraints. To avoid oscillations in the agents' behaviour, one can easily extend the approach to Reciprocal Velocity Obstacles (RVO) [9] and Hybrid Reciprocal Velocity Obstacles [6]. However, such oscillations did not cause problems in this setting and RVO performed likewise or worse compared to only the VO approach, therefore only VO was used.

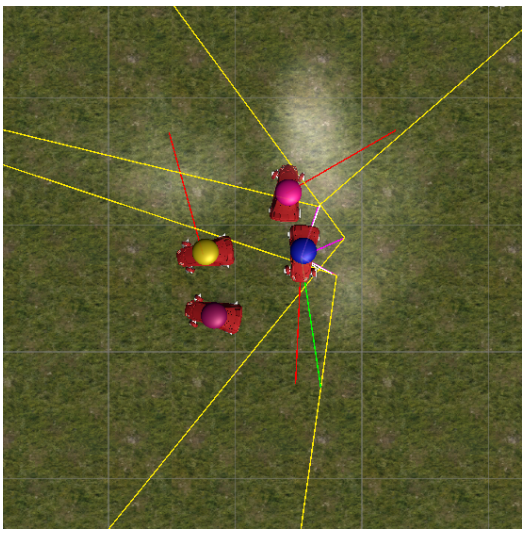

Figure 2: Here yellow vectors representing $VO_{blue|pink}$, $VO_{blue|yellow}$, and $VO_{blue|purple}$ (the yellow and purple car is reversing). Red vectors are the preferred velocities for each agent (here car-like robots instead of drone-like). We see that the blue cars preferred velocity lays inside $VO_{blue|purple}$ and needs to be recalculated. White vectors combined with the green vector represent candidates for a new velocity. The green candidate is the closest candidate to the preferred velocity, and chosen as the new collision-free velocity.

3.2 Problem 2: Drone soccer

In this second problem, a soccer team has to be controlled to win matches against the teams from the other groups. The teams are each composed of three drones able to move in all directions in the xy-plane, kick the ball with a limited velocity in the same plane and communicate between them. They are also able to access all the information about the positions and the velocities of each player on the field, of the ball and the two goals. The difficulty of this problem is to elaborate a complex strategy with the three drones of our team to score goals and to avoid taking some. To reach this objective we decided to use a behaviour tree (BT), which is a model that can handle the switching between different behaviours given the situation. The final BT corresponding to the strategy we implemented can be found in figure 3.

We first look if the ball is not very close to our ball (in the quarter of the field next to our goal) and if we are closer to the ball than the opponents are (*AttackFormation*). If we attack, the closest robot to the ball will either shoot if it can calculate a free-angle towards the opponent goal (*AngleOn-*

FreeGoal) or take a challenge if an enemy is coming (*EnemyArriving*), which means that our robot will go behind the ball on the line ball - own goal to protect our goal and elsewhere it will dribble towards the opponent goal by pushing the ball in direction of the centre of the enemy goal. The two other robots of our team will head towards the backup and the solution position (*Backup*). The backup is located behind (in direction of our goal) the ball on the same horizontal line as the latter. The solution is located a little bit behind the ball and more in the middle of the field. The backup and the solution positions while attacking can be observed in figure 4 (blue crosses). We optimize the assignment of which robot goes to the backup and which one goes to the solution by looking at the time needed to reach these positions.

In the case where we defend we first look if the ball is very close to our goal and if we can shoot the ball away from it (*CloseToBallAndFreeSpace*). In this case, we will look for the best shooting direction, towards the opponent goal and where the opponents are not located so that we do not make a direct pass to them. Elsewhere we look if the robot is the last defender (*IsLastDefender*), which means that there is only one of our robot between the ball and our goal. Then it needs to waste time to let enough time for our other robots to come back, so it will stay behind our goal and the ball and it will delay its challenge. In the case where several robots can defend and that the ball is in our half field (close to our goal, *InOurHalffield*) the closest robot to the ball is sent for a challenge, while the other two robots are assigned to the frontpost and the backpost positions (*Frontpost*). The frontpost is located at one third between the post of the same side of the ball and the ball. And the backpost is located at one third between the centre of our goal and the ball. The frontpost and the backpost can be observed in figure 5 (blue crosses in front of the players).

The shooting strategy is shown in figure 6. If the greatest free-angle towards the opponent goal is located between two players (right case) the robot shoots in the middle. Whereas if the greatest angle has a post as a limit (left side), we shoot in direction of a point located at $\frac{1}{2}(\frac{x}{G})^2$ of the post where x is the free distance on the goal where we can shoot and G the width of the goal. The radius of the robots is a function of the distance between the ball and the robot. The more the robot is far away from the ball, the more it has time to move to catch the ball, so its radius will be high. A robot close to the ball has little time to move so its radius is close to the radius of a drone.

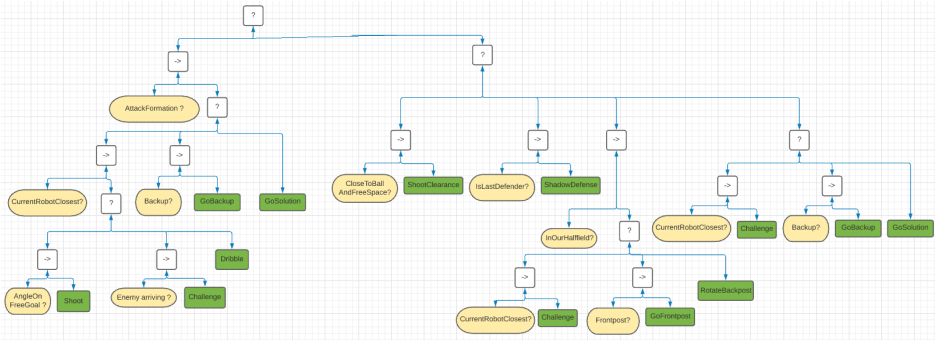

Figure 3: The behaviour tree controlling the different agents

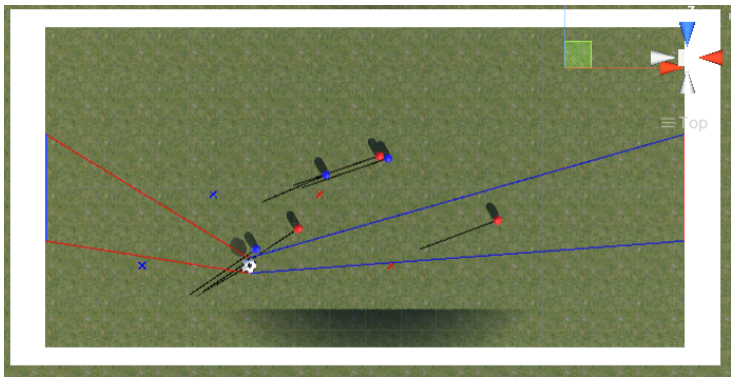

Figure 4: The backup and solution placements while attacking or defending

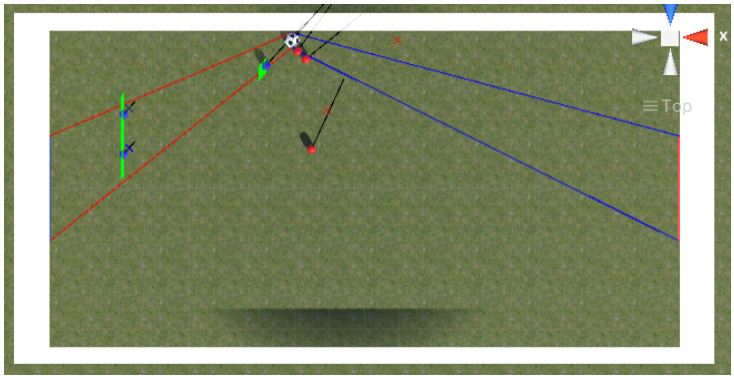

Figure 5: The frontpost and backpost placements

3.3 Implementation

4 Experimental results

In this section, the results of both representations are presented. Both implementations performed well, within total the 5th quickest time on the first

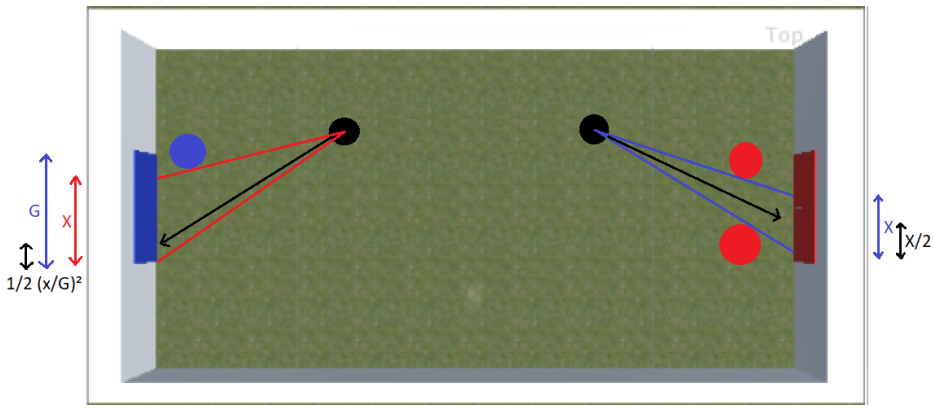

Figure 6: The shooting tactic

problem and a silver medal in the drone soccer league.

4.1 Problem 1: Collision avoidance results

For the space without static obstacles, this implementation almost manages to avoid collisions completely, and all agents manage to find their way to their goal position in under 30 seconds (second-best time among the groups). A snapshot of when the drones meet in the middle could be seen in the left figure of 7. For the map with a narrow junction in the middle, the space is more crowded and the agents have a harder time finding their way through. This leads to deadlocks where clusters of agents slowly are pushed towards an exit in the junction where the clusters lastly could dissolve. The right figure in 7 shows this behaviour, and also reveal a bug in the A* implementation which not return an optimal path. However, since the A* provided a descent path and due to time constraints, this bug was never solved. This implementation manages to solve this terrain in 137 second which is the 5th best time for this terrain. The results of the best groups could be seen in table 1.

4.1.1 Analysis of the results

Group 11 and Group 13 performed well on both terrains for Problem 1 and had a similar solution. Our approach was very similar to theirs; RVO for collision avoidance and A* for global path planning. The difference between our results and their results are also small for the open terrain. However, the difference on the terrain with a junction is bigger. This could probably partly be explained by our buggy A* algorithm not providing an optimal path. Group 4 went down a slightly different path, they solved the problem

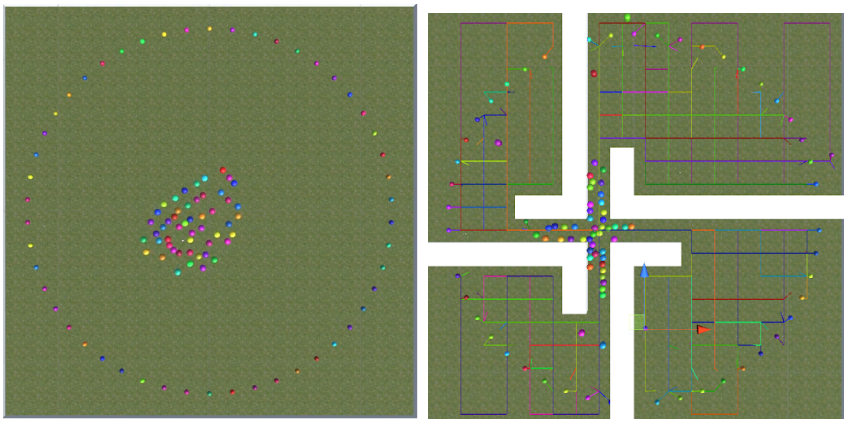

Figure 7: Left: Agents trying to find their way towards their goal while avoiding collisions in a open space.
Right: The more demanding terrain with a narrow junction

	Open Terrain (sec)	Terrain w/ Junction (sec)	Drone Soccer (points)
1st	Group 1 (20)	Group 11 (62)	Group 2 (56)
2nd	**Group 10 (29)**	Group 3 (96)	**Group 10 (55)**
3rd	Group 11 (34)	Group 4 (96)	Group 13 (54)
4th	Group 13 (37)	Group 13 (98)	Group 4 (51)
5th	Group 4 (43)	**Group 10 (137)**	Group 3 (42)

Table 1: The results for the best performing groups for the two terrains of Problem 1 as well as the soccer problem. The performances of the implementations in this work are highlighted in bold.

with an A* algorithm searching in 3D space, also taking the time dimension into account. As seen in table 1, this approach performs quite well.

4.2 Problem 2: The Drone Soccer League

The results are presented in the Table 2. We finished second in the soccer final game with one point less than the winning group 2. However, we performed the most wins (18) and the highest score difference (40) among all the teams.

4.2.1 Analysis of the results

The four leading teams were very close to each other in the final league. Every group decided to use a BT to control its soccer drone team. If we compare the solutions provided by the first four groups, one can first see that they

	Group	Points	Wins	Draws	Losses	Score difference
1st	2	56	17	5	2	37
2nd	**10**	**55**	**18**	**1**	**5**	**40**
3rd	13	54	17	3	4	39
4th	4	51	16	3	4	27

Table 2: The results of the best teams during the Drone Soccer final

are sophisticated. The BT implemented in this paper can be observed in Figure 3. A complex problem like this one needs to be solved with a complex solution. This can explain why complex strategy performed better. The main difference between our strategy, inspired by the common strategy among the best players of the E-sport game Rocket League, and the strategies of the three other leading groups is the absence of a goalie. In our strategy, when we attack, our three drones are moving forwards applying a lot of pressure on the opponents and when we defend our three drones are defending. The advantage of such a position is that the agents can very easily rotate so that there is always one robot that can go quickly on the ball. However, it presents two disadvantages. The first one is that some teams let a player in front of our goal waiting for a pass and since we had no goalie and that the switching between the attack and the defend formation took some time it was hard for us to defend such goals. The second one is due to one rule of the games: every time we scored, the ball respawned immediately in the middle of the field, while our three drones were attacking and thus standing on the wrong side of the field to defend our goal. This leads to some goals directly after we scored one. But as we can see, despite these two disadvantages we performed very well, showing that a goalie was not needed to win.

5 Summary and Conclusions

In this work, we have presented approaches to two interesting problems, collision avoidance between multiple agents and playing a soccer game. For the collision avoidance problem, we used velocity obstacles in combination with a PD-controller and an A* algorithm for global path planning. For the soccer game, we have presented a behaviour tree inspired by tactics used in the game Rocket League. These two solutions perform well and end up in the top 5 among the other groups in the course. The solutions are also flexible and could easily be transferred to other problems settings. However, they both have room for improvements.

5.1 Future improvements

Both solutions depend a lot on different parameters. A rigid parameter search would probably bump up the performance of both solutions.

The solution for problem 1 could be improved in several ways. The solution could be extended to take the kinematic constraints of the agents into account, which would increase the accuracy when calculating a new velocity. Another improvement could be to introduce priorities among the agents to avoid deadlocks, which some of the other groups did in the course. However, the first improvement would probably be to use a functioning A* algorithm, which would speed up the solution.

The behaviour tree for problem 2 could be improved with a more complex version of it. It is actually too simple, and both the functions (like when we decide if we are in attack or defence) could be complicated and the number of behaviours (we could add a passing behaviour between the three agents for example). We also do not look yet when the ball is in the air, leading to some fails from our team at the start of the match. We could improve this by calculating where and when the ball will fall on the ground. The shooting strategy could also be improved to take more into account the current velocity of the ball: we should try to find a good angle close to the current velocity of the ball because shooting in an opposite direction will lead to a very slow ball which can be stopped very easily. The drones are also a bit struggling to catch the ball, so it could be possible to calculate more precisely how to catch it. It would also be possible to try some non-hard coded behaviour trees by using genetic programming or reinforcement learning in addition to the BT according to the corresponding parts of the book [1].

References

[1] Michele Colledanchise and Petter Ögren. Behavior trees in robotics and AI: an introduction. *CoRR*, abs/1709.00084, 2017.

[2] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[4] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987.

[5] R.G.Dromey. Genetic Software Engineering - Simplifying Design Using Requirements Integration. 2001.

[6] J. Snape, J. v. d. Berg, S. J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

[7] Kok K Tan. *Advances in PID Control*. Advances in PID Control. 1st ed. 1999.. edition, 1999.

[8] Ruoyu Tan and Manish Kumar. Proportional navigation (pn) based tracking of ground targets by quadrotor uavs. volume 1, 10 2013.

[9] J van den Berg, Ming Lin, and D Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.