Anti-Collision Implementing Velocity Obstacle Method

Luis Mercado, Sabrina Fong, Pauline Mae Cuaresma
ECE 163 Fall 2020
December 17

I. Introduction on VO and Concept

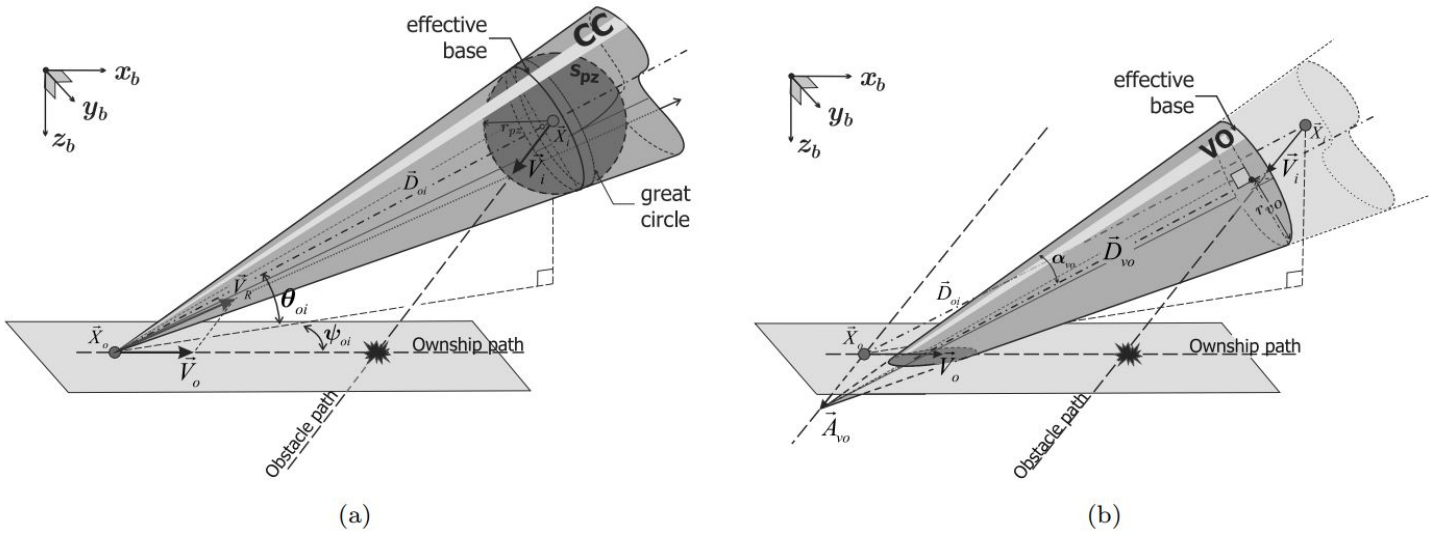


Figure 1: Illustrations of the collision cone (CC) and velocity obstacle cone (VOC) with the parameters needed to obtain each of the cones [1]

In the case that an unmanned aerial vehicle (UAV) encounters dynamic or static obstacles during flights it is essential for the UAV to be able to safely dodge these obstacles to prevent collision. For communication network systems operating multiple UAVs, it may be impossible for two UAVs to exchange information as the distance between them may be beyond their communication range. Therefore, it is beneficial for UAVs to implement an autonomous Conflict Detection & Resolution (CD & R) algorithm to ensure safety during flight.

Velocity Obstacle (VO) enables the UAV to autonomously determine possible safe maneuvers to avoid collision. VO methods and algorithms typically implement sensors on board to collect information from their surroundings. The distance from itself to the obstacle, d_{oi}, the owner UAV velocity, v_{own}, the obstacle velocity, v_{ob}, and the radius of the obstacle, r_{ob} , are other important parameters required by the UAV for VO. A collision cone (CC) is created between the owner UAV and the obstacle, with the apex of the CC being at the origin of the owner UAV. The CC, which can be drawn at every encounter with an obstacle, collects all relative velocity vectors between the two vehicles that intersect the protected zone of the obstacle (S_{pz}). S_{pz} is a threshold area around the obstacle. The radius of the S_{pz} is typically the summation of the vehicle's effective semi-spans. If the velocity of the owner UAV is included into the CC, then it can be determined that a collision would

occur. To avoid collision, the owner UAV needs to change its velocity in a way that it is not included in the CC set.

The VO-method takes into account both the obstacle's velocity and position. To do this, the CC is translated by the velocity of the obstacle. The newly translated CC is referred to as the Velocity obstacle set (VO). A collision between the two objects would occur if, and only if, the velocity vector is included in the VO cone. The VO cone is defined by three parameters: the position of its apex A_{vo}, the length and orientation of the axis D_{vo}, and the radius of S_{pz}. These three parameters are mathematically express by the following equations:

$$d_{vo} = \frac{d_{oi}^2 - r_{pz}^2}{d_{oi}} \qquad r_{vo} = r_{pz}\left(\frac{\sqrt{d_{oi}^2 - r_{pz}^2}}{d_{oi}}\right), \qquad a_{vo} = arctan\left(\frac{r_{vo}}{d_{vo}}\right)$$

$$A_{vo} = V_i, \quad \text{and} \quad D_{vo} = \left[\left[cos\theta_{oi} cos\psi_{oi}\right], \left[cos\theta_{oi} sin\psi_{oi}\right], \left[sin\theta_{oi}\right]\right] d_{vo}$$

$$\theta_{oi} = arcsin\left(\frac{d_{oiz}}{d_{oi}}\right), \quad \psi_{oi} = arctan\left(\frac{d_{oiy}}{d_{oix}}\right)$$

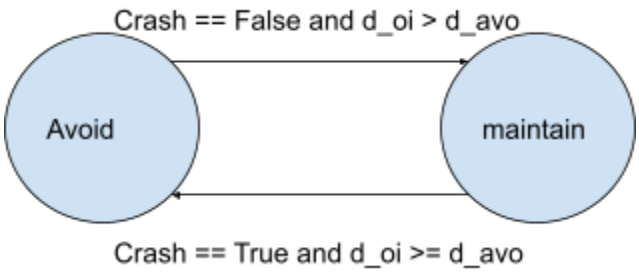
For the owner to start calculating an avoidance vector, the velocity vector needs to be included into the VO cone, and the distance between the two objects need to be below avoidance starting distance/ critical avoidance distance (d_{avo}). The owner UAV can calculate the avoidance vector, the owner UAV needs to update its velocity to a point outside every relevant VO set, into the set of avoidance velocities (V_{avo}). The avoidance panes (P_φ), are defined as any plane in which the ownership V_o lies. The avoidance is assumed to be conducted in one of these planes around the X-axis. The avoidance plane is chosen by weighing the danger of each plane based on the shape of the VO section.

In order to implement the VO-method in a three dimensional space, two parameters are needed: the avoidance distance (d_{avo}) and the turning rate (ω_{avo}). The turning rate is the required maneuverability for the available avoidable distance. Using the parameters, the UAV can determine how to use the remaining space to maneuver the aircraft to independently determine anti-collision routes. Our project attempted to implement the VO-method with a moving object (owner UAV), and a static obstacle.

II. Methods

Due to the unexpected complexities of this project our team decided not to implement parts of the VO algorithm to autonomously determine necessary parameters. Therefore our ***code partially implements the velocity obstacle algorithm*** for UAV anti-collision under the condition that the obstacle is static.

The team's approach to implement this algorithm can be described with a simple state machine.



While the owner UAV is flying around, if the velocity is not included into the VO, then it would behave normally. If the velocity vector, of the owner UAV, is included in the VO, and the distance between the objects are closer than the designated avoidance distance, then the owner UAV would go into the avoid state. Here the owner UAV would determine to either fly above or below the obstacle.

Velocity obstacles is a complicated topic and requires assumptions in order to make its implementation possible. The specific assumptions the group made to simplify the problem were:

1. The velocity obstacle cone is abstained by translating the collision cone by the velocity vector of the obstacle. Our obstacle contains no speed, so we assume that the collision cone is the velocity obstacle cone for our case. Thus, $A_{vo} = V_i = 0$.
2. We assume that the coordinates (NED) of the second UAV (obstacle) is known already as well.
3. The distance of the owner UAV to the obstacle is a predetermined distance. This was done due to lack of time, and experience with the VO-method.
4. Due to there only being one static obstacle in an open field, we assume that the owner UAV does not need to consider which avoidance plane to move in.
5. The turning rate for the owner UAV is not implemented in this project. Instead we implemented a random number (0 or 1), which tells the owner UAV to travel either above or below the obstacle.

Many of these assumptions are due to the team's lack of experience and time to implement the VO-method.

We made several modifications to the Vehicle Aerodynamics Model and Vehicle Closed Loop Control and Vehicle Display files in order to implement the velocity obstacles. To draw the second UAV that acted as the obstacle, another instantiation of a UAV was added to the Vehicle Display. A new UAV object was instantiated in the init file in the Vehicle Aerodynamics Model. This new UAV object will be used as the static object that the velocity obstacle would detect and the UAV will avoid. The new UAV object would have its coordinates already predetermined, instead of creating a dynamic object, because creating a dynamic object would cause a more complex problem. In order to implement the velocity obstacle algorithm, three new functions were created: distance, VOupdate, and avoidance.

The distance function is a simple function that calculates the north, east, and down distance between the origin of the UAV, and the static UAV obstacle, and returns the calculated distance value, which is the hypotenuse of the differences. This function is called every time VOupdate is called.

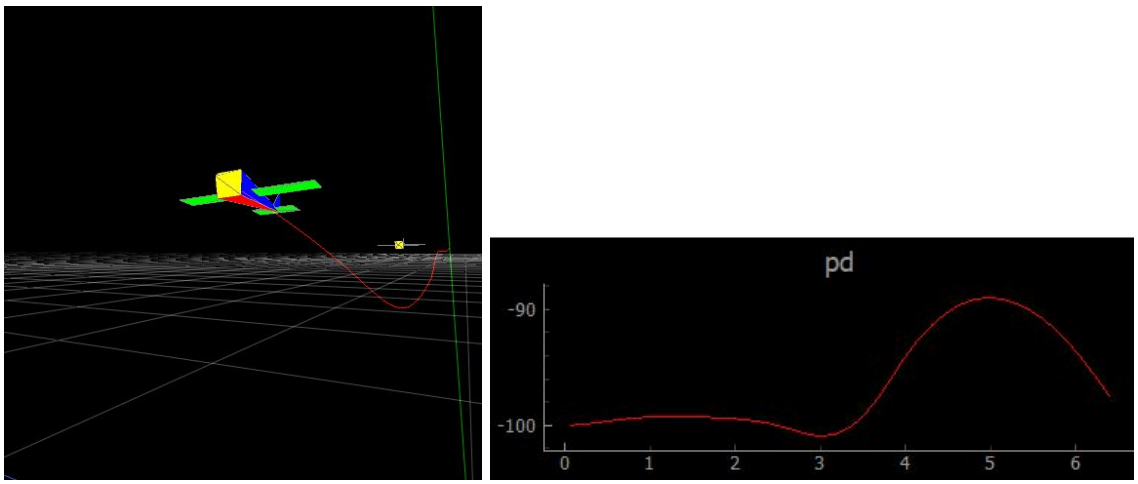
The VOupdate is the main function in implementing the velocity obstacle algorithm. The collision cone set can be drawn for every encounter. When creating the CC, the distance between the owner UAV and the obstacle is calculated first. The elevation angle, and the azimuth angle for the cone are produced with the components from the distance vector, between the two objects. The radius of S_{pz} was set to the wingspan of the UAV. The CC is determined by the mathematical equations stated above. In order to determine if the owner UAV velocity is included in the CC, a boolean variable was included. The boolean variable (self.crash) would turn to True, if the owner UAV is an element of the CC cone, and False if it is not. The check that was implemented is as follows:

$$V_o \, \varepsilon \, CC \;\Leftrightarrow\; \{\frac{[V_o]\cdot D_{vo}}{|V_o| d_{vo}} > cos\alpha_{vo} \; and \; d_{oi} < d_{avo}\}$$

This check ensures the UAV initiates an evasive maneuver when the angle of the VO cone is large enough and the UAV is within its avoidance starting distance to elicit an evasion. If the check fails, then the boolean variable stays false and the owner UAV does nothing. If the check passes, then the boolean variable is set to True. If the self.crash is True, then the owner UAV would then go into avoidance.

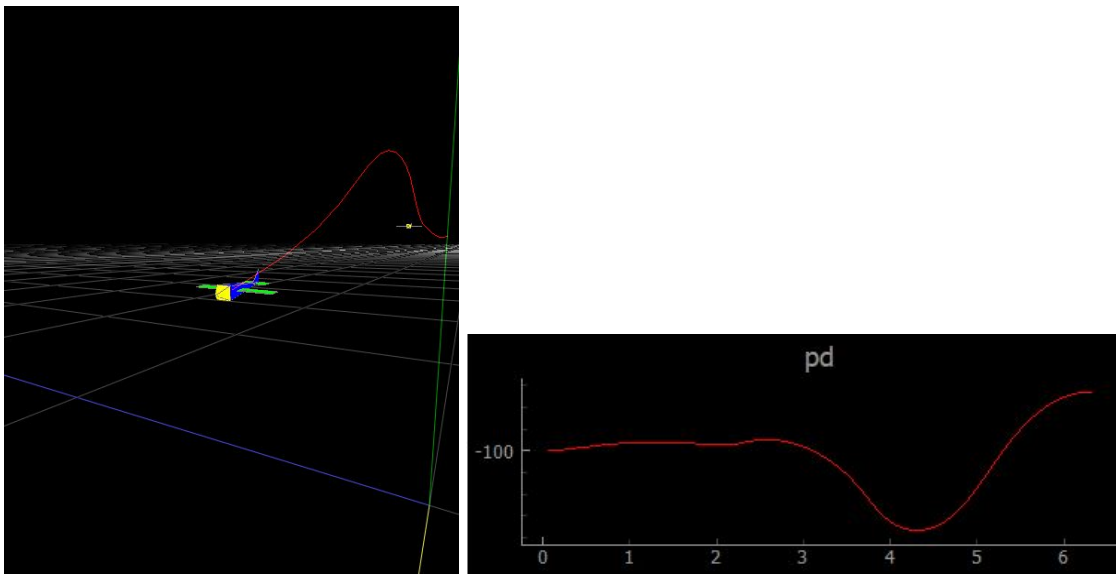
To translate the crash signal into changes in the states and movement of the UAV, the closed loop control module was altered. The methods for ascending and descending when the UAV was out of the hold zone were replicated and used to ascend or descend the UAV when the crash signal was true. The UAV moved based on the avoidance direction. If the avoidance direction was up, the UAV ascended. If the direction was down, the UAV descended. Because the avoidance direction was determined using random and the avoidance direction function was called at each iteration of VOupdate, which was at every call of the Update function of the Vehicle Aerodynamics Model, the initial direction was saved (as self.prevDir) at the first appearance of the crash signal. It was reused in subsequent appearances of the crash signal for the same obstacle. The procedure ensured that the UAV moved in the single avoidance direction for the same obstacle. The initial direction is reset after the crash signal is False.

III. Results



(a) (b)

Figure 2: The UAV goes in a straight path until it reaches a critical distance for avoidance. It then undergoes the descending avoidance procedure. (a) Down-direction avoidance and (b) an example graph of the UAV's elevation change (bump) that occurs during the avoidance procedure. For both Figure 2a and 2b, d_{avo} = 50 m and v_{own} = 25 m/s. There was no wind.



(a) (b)

Figure 3: The UAV goes in a straight path until it reaches a critical distance for avoidance. It then undergoes the ascending avoidance procedure. (a) Up-direction avoidance and (b) an

example graph of the UAV's elevation change (dip) that occurs during the avoidance procedure. For both Figure 3a and 3b, d_{avo} = 50 m and v_{own} = 25 m/s. There was no wind.

As mentioned in the Methods section, our team only has partial implementation of the Velocity Obstacle algorithm due to its complexities in executing the code and due to the limitations we had with the simulation. For example, the fact that simulated UAV did not have actual sensors on board to detect how far and when an object is in its collision cone was one major limitation to properly develop a VO algorithm. To bypass this, the team had to hard code the coordinate location of the obstacle UAV while also making the object static. The location of the static object was therefore known to the UAV rather than autonomously calculated and accounted for by the VO algorithm. The obstacle UAV was given the coordinates, Obstacle = (99.2, 0, -100). Other parameters that the simulation did not implement was determining the radius of the obstacle and determining the obstacle's airspeed velocity. These parameters were also hardcoded into the program due to the fact that programming sensors on the UAV would be necessary to obtain these parameters (that in itself would be another final project). The radius of the obstacle was set to be equal to the wingspan of the UAV, r_{ob} = VPC.b. We decided to make the radius equal to the UAV wingspan to account for the possibility of being less than the length of the UAV. In general, based on the VO theory, it is safer for the UAV to assume the obstacle has a wider radius than a smaller one. Since the UAV is static, its velocity vector, v_{ob}, was set to be the zero vector. With these parameters, for each set of vehicle states at a given increment dT, the UAV was autonomously able to calculate its obstacle velocity cone (relative to the UAV) and its set of obstacle velocities that would lead to collision ime. It was also able to autonomously determine if its current airspeed would lead to collision and autonomously maneuver itself to an escape route. UAV flight in the simulation was *intended not to include wind.*

In classic VO algorithms, once a UAV determines the potential of a collision, it will also calculate a potential anti-collision velocity vectors needed to safely escape collision given the limited space left between itself and the obstacle. With this, the UAV can also determine the appropriate turning rate to ensure that it safely maneuvers itself. Our group was not able to implement the mathematics to determine avoidance planes, shown in Figure 4, that would have helped us with programming the UAV to autonomously calculate potential escape velocities and corresponding turning rates. To bypass this, we made a function in the VAM to detect a potential crash using the velocity obstacle cone. If a potential crash is detected, the UAV is programmed to go up or down (by adjusting its altitude) to avoid collision. Once a crash is no longer detected, the UAV can continue holding its current position. Classic VO algorithms also typically autonomously calculate the necessary distance from the obstacle it needs, given the airspeed and turning rate, to successfully prevent collision. Since we were not able to implement avoidance planes to our project, this parameter was also hardcoded to be d_{avo} = 50m. Our team discovered that d_{avo} > 100 m resulted in our UAV oscillating. *With the UAV unable to autonomously determine an escape velocity in our partial implementation of VO in 3D, the airspeed of the UAV should remain at 25 m/s or below to ensure that our VO algorithm is successful*.

Despite several limitations and the dense number of papers we had to read, as shown in Figure 2 and Figure 3 the UAV ***successfully*** avoided collision with the static obstacle with the partial VO logic we programmed! The code was run on Chapter7.py. *A video of our project working is included in our group repo.*

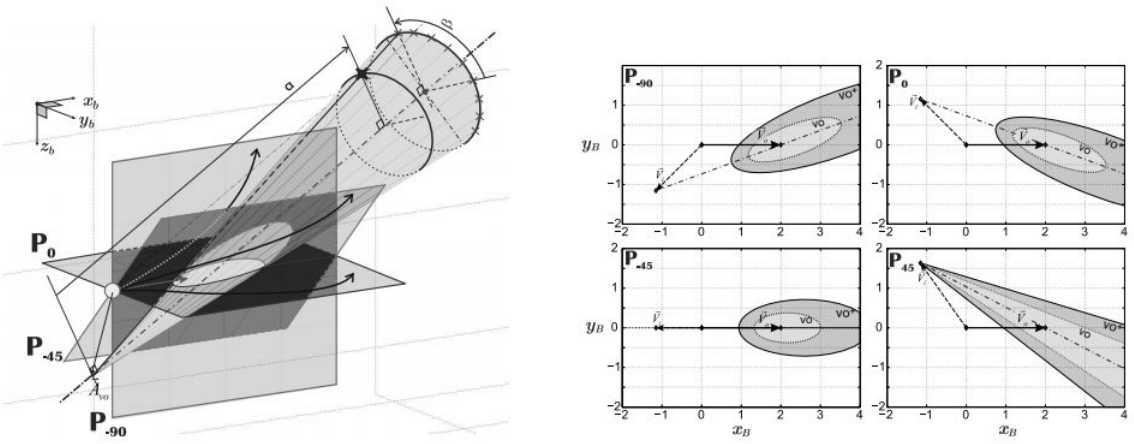


Figure 4: Illustrations of avoidance planes for velocity obstacle escape maneuvering algorithms [1].

IV. Discussion

There were several challenges to the project. Initially, our team was only able to find good documentation for VO anti-collision in 2D. It took some time and digging to find ways to get access to research papers on 3D implementation of VO. However one of the biggest hurdles we had was trying to understand velocity obstacles and all the mathematics and concepts required for its implementation. The reviewed papers were dense and even when we found an existing explanation of how to execute VO in 3D, understanding it was extremely difficult. We made several attempts to try to implement certain aspects of VO in our project, however, many functions failed or were too complicated for us to determine how to properly implement. In regards to the process leading up to a working implementation, we made small changes in many places. Some of the changes stayed, like changing the method for determining the distance from using math.sqrt() to math.hypot(). Others, like putting VOupdate in the Sensors Model module, were discarded. Finding the source of changes in the UAV's movement was also a challenge and required tracking of which and why certain conditional statements were implemented instead of the expected ones.

The project pushed the team to learn about velocity obstacles, mainly from the existing work. We learned the theory behind velocity obstacles and why this topic is important to expanding the capabilities of a UAV in the real world. We also learned how to use existing code from the class labs to build the algorithm.

Based on the results, the project has several areas of improvement. One of the areas is an algorithm for stopping a crash signal from occurring after the UAV has passed the obstacle.

Another area is adjusting the critical avoidance distance and the timing of the crash signal so that the UAV does not stray as far from its original path. Last but not least, an algorithm for figuring out the direction of the avoidance so that it is not randomly determined.

This project has many areas to explore. For one, more dynamic and static obstacles and even other UAVs with the same obstacle avoidance algorithm can be added to the simulation. The UAVs can be programmed to fly in a pattern without collision with one another. To improve the avoidance model, the project can have more complex procedures like reciprocal vehicle obstacle (RVO) and hybrid reciprocal vehicle obstacle. Finally, movement in all directions, i.e. up, down, left, right, and everything in between, would improve the algorithm's ability to move in the real world, where many obstacles and tight, winding spaces exist.

V. Conclusion & Future

The group faced many challenges with this project but at its completion, it was able to simulate a velocity obstacle using a static obstacle with fair results. The UAV could fly up or down to avoid a static obstacle that had parameters, like its fixed radius and position, known to the UAV. The code took many hours of tinkering and even more hours of understanding the theory behind the problem it was addressing. The challenges made us appreciate the complex UAV autonomous algorithms that can be implemented and peaked our interest in UAV other implementations to UAV programming. Although many hours were put into completing what looks like a simple implementation, the success with the project encourages the group to explore more possibilities and improvements, like those mentioned in the future work.

For future work, the VO algorithm can be expanded. As mentioned in the results, several aspects of our program excluded classic VO algorithms due to our limitations or the complexities behind obtaining the necessary parameters autonomously. Therefore, our VO algorithm can be improved to be more movable (have more degrees of freedom for avoidance), dynamic, obstacle, and autonomous. For example, buffer velocity sets can be implemented to further improve the avoidance capability of the ownhip UAV. The addition of avoidance planes, to decrease the computation strain on the simulation. Instead of deriving all possible avoidance, the method would only focus on a finite number of avoidance planes. To improve the smoothness of turning (avoiding), the turning rate can be implemented into this method. Finally rather than avoiding a single static UAV, our program can be improved to avoid multiple static UAVs (given that their coordinates are known or a sensor is implemented to detect them on board) while including wind in the simulation.

VI. References

[1] Jenie, Yazdi & Van Kampen, Erik-Jan & De Visser, Coen & Ellerbroek, Joost & Hoekstra, Jacco. (2016). Three-Dimensional Velocity Obstacle Method for Uncoordinated Avoidance Maneuvers of Unmanned Aerial Vehicles. Journal of Guidance, Control, and Dynamics. 1-12. 10.2514/1.G001715.