Introduction

Introduction to report goes here: Talk about motivations and contributions

Artificial Potential Field Method

The artificial potential field (APF) method [OussamaKhatib] is perhaps one of the most popular methods for collision avoidance in robotics and has been used for mobile robots and manipulators alike. Here, the obstacles and goal are treated as point sources from which potential fields are implicitly emitted. The goal emits an attractive field, allowing the robot to be drawn towards its target, whilst the obstacles emit a repulsive field. In particular, these are given in [MultiPointAPF] as

$$U_{att}(\overrightarrow{x_r}) = \frac{1}{2}k\left(d_{goal}(\overrightarrow{x_r}) + u\right)^2$$

$$U_{rep_{(o)}}(\overrightarrow{x_r}) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{d_o(\overrightarrow{x_r})} - \frac{1}{d_{rep}}\right)^2 d_{goal}(\overrightarrow{x_r}), & \text{if } d_o(\overrightarrow{x_r}) \leq d_{rep} \\ 0 & \text{otherwise} \end{cases}$$

where $\overrightarrow{x_r}$ is a position in the environment, k and η are coefficients on the attractive and repulsive fields respectively denoting their relative priority (Section …). d_{goal} and d_{obs} denote the distance from the $\overrightarrow{x_r}$ to the goal and obstacle respectively. u is the forward velocity of the robot and d_{rep} is the minimum distance from the obstacle before it is considered by the planner. The total potential $U(\overrightarrow{x_r})$ at a given point is given by the sum of the attractive and all repulsive forces. Given N obstacles, this is formalised as

$$U(\overrightarrow{x_r}) = U_{att}(\overrightarrow{x_r}) + \sum_{o=1}^{N} U_{rep_{(o)}}(\overrightarrow{x_r})$$

Note: The attractive potential in the method described in this section did not make use of the u term in (Attractive Potential Equation). This is since, as will be discussed, the UAV applied in this thesis may move in any direction and so no ‘forward’ direction is defined. The remaining equations are unchanged so it is sufficient to view the u term as set to 0.

At its core, the APF method simply models the environment with the potential function $U(\overrightarrow{x_r})$. The ultimate aim of the planner is to arrive at the global minimum (i.e. the goal location). A straightforward way to visualise this process is to consider (in the case that the robot can only move along a plane) that the potential field is a surface embedded in a three-dimensional space. The attractive potential can be pictured as a bowl-like formation around the goal while the repulsive field caused by the obstacles form a hill-like formation. An example of this is shown in Figure … The robot then acts as a ball placed on this surface which steps towards the minimum point on this surface while avoiding the obstacles.

There are a number of advantages to this method which resulted in its consideration for this study. The first is its intuitive and simplistic approach - it is straightforward to understand and communicate the process by which the planner operates. More importantly, however, is the versatility of the potential function itself. Since it is a defined property, it is straightforward to adapt the potential fields to accommodate different environments and heuristics. For instance

[ProbMapAPF] and [VFF] adapt the repulsive field to account for the uncertainty in the robot's perception of the environment. This allows the robot to avoid areas where it is more certain an obstacle lies.

When minimising $U(\overrightarrow{x_r})$ through a traditional optimiser (such as gradient descent etc.), it is required that the gradient of the potential field be determined at the robot's location. This can become an expensive procedure, limiting the real-time applicability of the method. However, it is not strictly necessary to determine this gradient to move the robot. Instead [Multipoint UAV] suggest that it would be more efficient to simply consider the value of $U(\overrightarrow{x_r})$ at discrete points surrounding the robot. The robot can then be directed towards the point with the minimum potential value. [Multipoint UAV] apply this methodology to a planar problem with an autonomous underwater vehicle (AUV) as depicted in Figure … This overcomes the aforementioned drawback of optimisation since it does not require the determination of complex Jacobians. Furthermore, this technique allows for the consideration of vehicle dynamics. For instance, the arc can be constrained so that only directions which are possible by the robot are considered. An example of this scenario is shown in Figure …

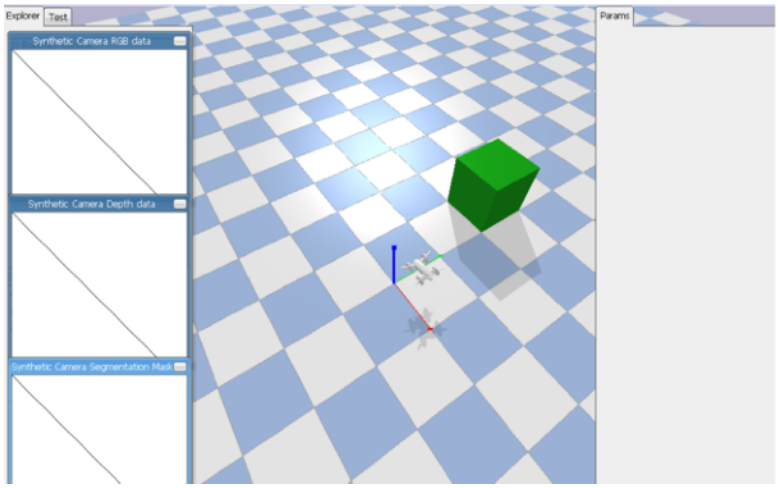
This thesis begins by applying the Multipoint APF method to a planar problem of a drone avoiding obstacles in a planar environment. This is followed in Section … by extending the method to be applied in a three-dimensional problem.

MultiPoint APF method

The original multipoint method described in [MultiPoint AUV] presumed that the AUV would consistently move in a 'forward' direction (forward defined as moving towards the goal) and so it was only required that an arc be formulated in front of the robot. However, in the case of a drone avoiding obstacles in an unknown, dynamic environment, it will frequently be required that the drone withdraw away from the goal in order to avoid an obstacle. As such, the arc was extended to form an entire circle with a known radius around the robot. It was then discretised into eight equiangular points where the potential field could be determined at each time step (defined in simulation as every 1/240 s). The drone would then choose the point with the lowest potential value as its setpoint (its goal location for the next time step) and so would start moving towards this point.

The radius of this circle became an important factor to consider since it would determine the speed of the drone. A larger radius, therefore, caused the drone to overshoot past the global goal location or to exhibit more jerky motion as its required velocity would change dramatically when the setpoint changed. On the contrary, a shorter radius would cause the drone to move slowly and so may not be able to execute critical avoidance strategies within the required timeframe. This became a significant problem in situations as shown in Figure … where the obstacle was either initially very close to the drone or was moving at a high speed. The radius was initially chosen to be a constant value to ensure the stability of the controller (i.e. avoid the jerky motion which results from the high velocities) but can be treated as a variable to allow for velocity adjustment based on the situation in question.

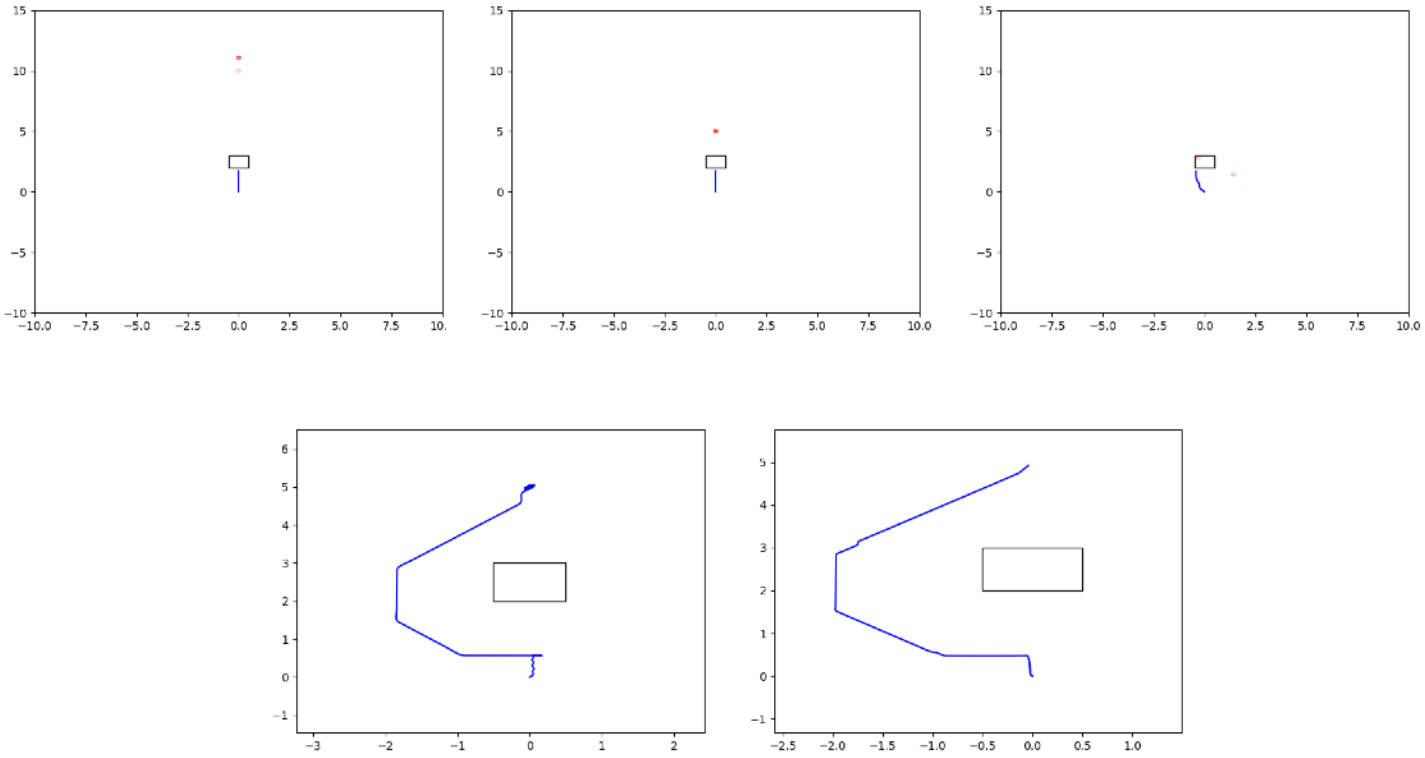
PARAMETER DETERMINATION



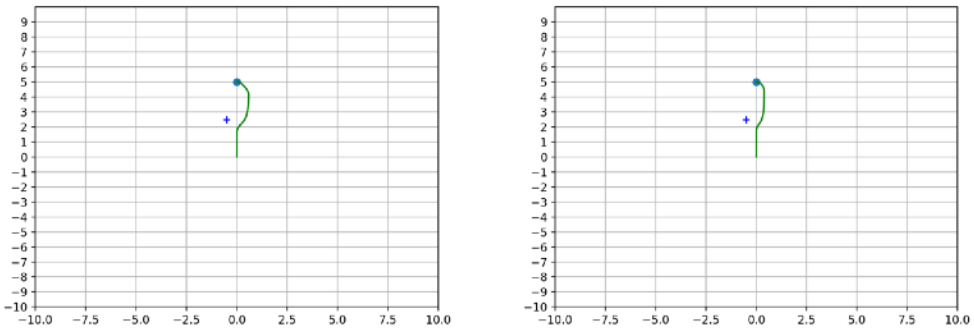
The full set of parameters required to define the path planner at this stage is shown in Table …. The method by which these are determined follows.

Term	Description	Final Chosen value
r	Radius of circle around robot	0.2m
d_{rep}	Repulsive field threshold distance	40m
A	Multiplicative factor between k and η	100
η	Repulsive field coefficient	1

The method was first tested with one static obstacle as shown in Figure … to assess for an optimal radius value. It was assumed that the robot has complete knowledge of the position of the obstacle and is in a free environment. Note that the performance of this method is highly dependent on the values of the constants k, η and d_{rep}. For the purpose of this examination these values were set as k = 10, η = 1, d_{rep} = 10. Figure … shows the importance of the role that the radius value plays since it determines the planner's capability to interact successfully with the controller to reach the goal position. Too high a radius (e.g. $R = 10m$) would result in potentials being evaluated too far from the robot to accurately represent obstacles closer than this distance (see Figure …a) Furthermore, a higher radius may lead to oscillatory behaviour close to obstacles and the goal since the arc point consistently overshoots the minima. A value of 0.2m was eventually chosen as the radius as, at this radius, the planner was consistently able to track the setpoint without overshooting at the goal.



A preliminary study was then conducted to determine optimal values of η and k for this same scenario. d_{rep} determines not only the strength of the repulsive field but also at what distance



Figures showing the path (green line) taken by the drone to avoid an obstacle (black cross). a) (Left) d_{rep} = 1, η = 1 b) (Right) d_{rep} = 40, η = 0.01. It can be seen that the trajectories followed by these settings are almost identical with the avoidance manoeuvre beginning at the same point

from the obstacle the robot considers the repulsive field from the obstacle. Since the strength of the field can also be controlled by η (Figure …), d_{rep} can be chosen to match the range of the onboard sensor, so that the repulsive field from the obstacle is considered as soon as the obstacle is detected. In this case, the sensor was arbitrarily chosen to be the LIDAR-Lite 3 Laser Rangefinder by Garmin [LIDAR] which has a range of 40m. Hence, d_{rep} = 40m.

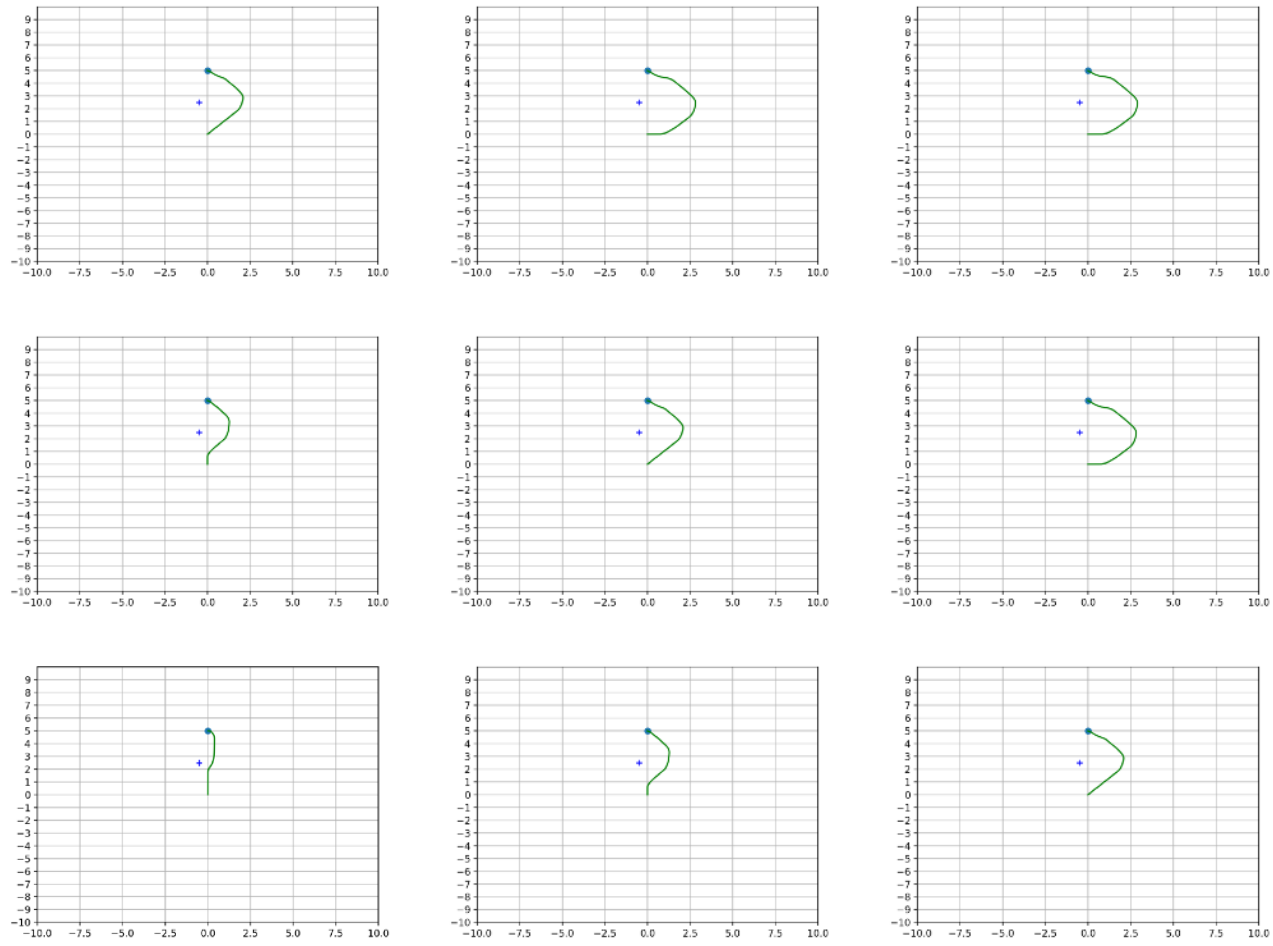
k and η can then be tuned to appropriately balance the attractive and repulsive fields. Increasing k increases the propensity to move towards the goal whilst increasing η results in an increased preference to move away from nearby obstacles. Figure … shows the paths followed by the drone using a range of values and were evaluated by considering the time taken to complete the path and the distance of closest approach of the drone to the obstacles. The closest approach gives a robust assessment of how far wide the path found by the planner is; a smaller distance allows for the robot to avoid the obstacles more narrowly which is expected to prove advantageous in narrower spaces. The results of this test are shown in Tables … and …

Time to completion (s)	$\eta = 0.1$	$\eta = 1$	$\eta = 10$
$k = 0.1$	16.22	18.58	18.83
$k = 1$	15.07	16.22	18.58
$k = 10$	13.91	15.07	16.22

Table showing the distance of closest approach from the drone to the obstacle placed at (-0.5, 2.5) when attempting to reach a goal location of (0, 5).

Distance of Closest Approach (m)	$\eta = 0.1$	$\eta = 1$	$\eta = 10$
$k = 0.1$	0.759	1.14	1.19
$k = 1$	0.402	0.759	1.14
$k = 10$	0.12	0.402	0.759

Table showing the distance of closest approach from the drone to the obstacle placed at (-0.5, 2.5) when attempting to reach a goal location of (0, 5).



Figures showing the trajectories (green line) taken by a drone to avoid an obstacle (black cross). From top to bottom k = 0.1, 1, 10 and from left to right η = 0.1, 1, 10. It can be seen that, where the ratio of k:η is the same, the same trajectory is followed. Table … verifies that a ratio of 100 (bottom left) shows the strongest results for avoidance manoeuvre as the robot does not stray too further than necessary from the obstacle, producing the smoothest output and lowest path completion time.

It can be seen from these tables that, rather than the values of k and η, the ratio between the two parameters which determines the equiangular arc points chosen by the Multipoint path planner; all scenarios where k and η were in the same ratio produced identical paths. This is to be expected since the Multipoint method simply takes the arc point which minimises the potential. As such, only relative importance has an effect rather than the values themselves. This crucially decreases the degrees of freedom of the planner's parameters by constraining their interdependence. With the additional constraints that d_{rep} = 40 and r = 0.2m, the tuning narrows down to selecting

$$A = \frac{k}{\eta}$$

Whilst Tables … show that an optimal setting is A = 100, it is important to note that this was only tested on the particular case where one obstacle lies at (-0.5, 2.5) with a goal set at (0, 5). To test the validity of this setting, the method was tested in a number of scenarios in which the potential field method has been known to fail [VFF]:

I. Obstacle lying directly in the path between the robot and the goal

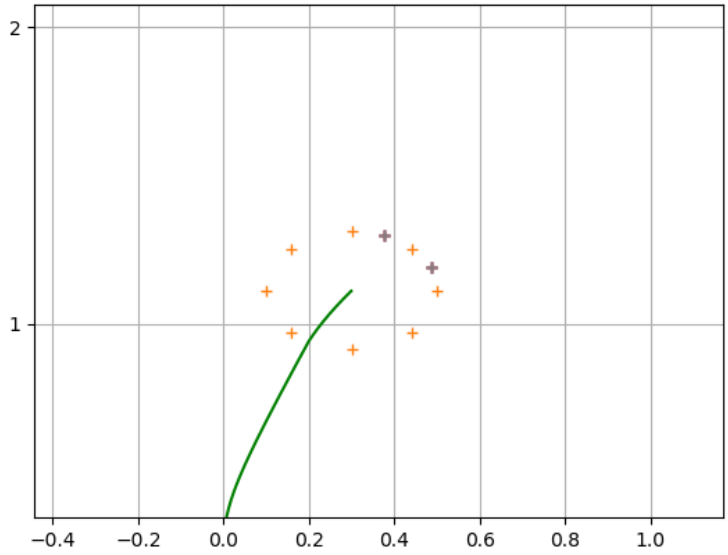
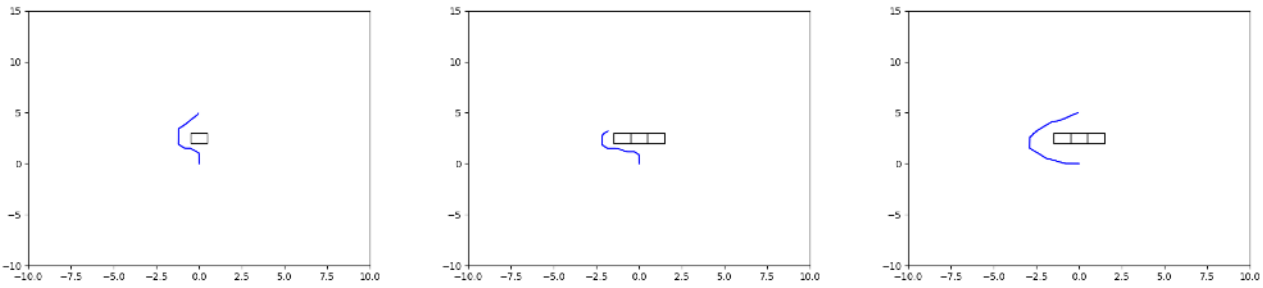


Figure showing two additional arcpoints (magenta) subtending an angle of $\pm\frac{\pi}{8}$ *rad from the previous arc point chosen by the planner. This gives the planner additional points to choose from, increasing the potential smoothness of the path.*

II. Obstacle moving towards the robot

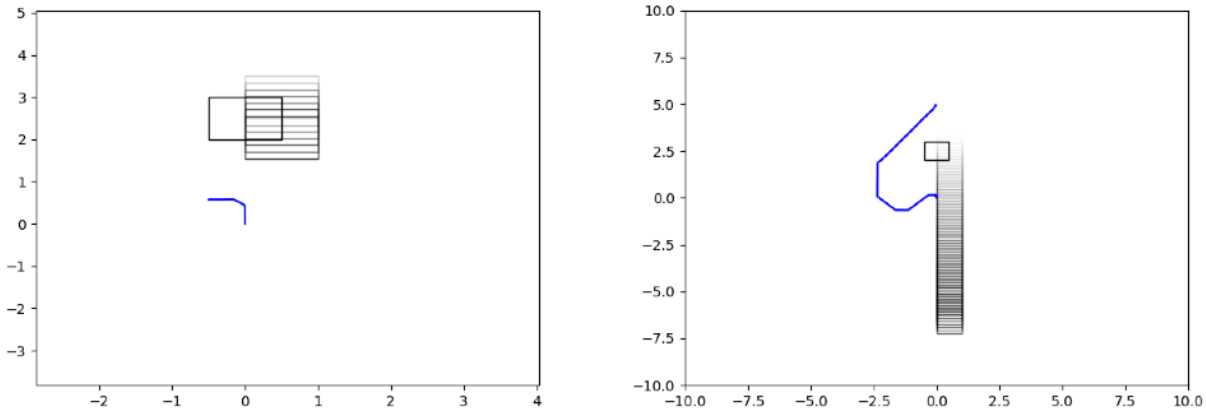
These situations can be seen in Figures … and … respectively. It was found that the setting with $A = 100$ was able to narrowly avoid a small obstacle. However, when multiple obstacles were placed next to one another (Figure …), the planner was not able to design a path to move around the obstacles and so consistently resulted in collisions. Similarly when testing the second scenario, the planner did not choose to move away from the moving obstacle early enough to avoid a collision. This is to be expected since, with A = 100, the tendency to move towards the goal vastly outweighs the tendency to avoid obstacles. In these situations, it was found that lower



a) Figure showing the trajectory (blue line) taken by a quadcopter using the Multipoint APF method with A = 100 to avoid an obstacle (black square) lying directly in its path. b) At A = 100, the planner is not able to sufficiently direct the quadcopter around the obstacles, indicated by the fact that the blue line terminates before reaching goal position of (0, 5). c) At A = 10, a suitable trajectory was found.

values of A (e.g. A = 10) would allow for safe passage around the obstacle since, in the first case, the robot would follow a wider arc (Figure …) and in the latter case, the avoidance manoeuvre would begin earlier, giving enough time for the drone to move out of the way.

However, as seen in Figures …, lowering A results in extremely wide arcs or decreased smoothness in the path. A large contributor to this problem is that the arcpoints are significantly spread apart (with an angle of $\frac{\pi}{4}$ rad between each point). This limits the choices of directions that

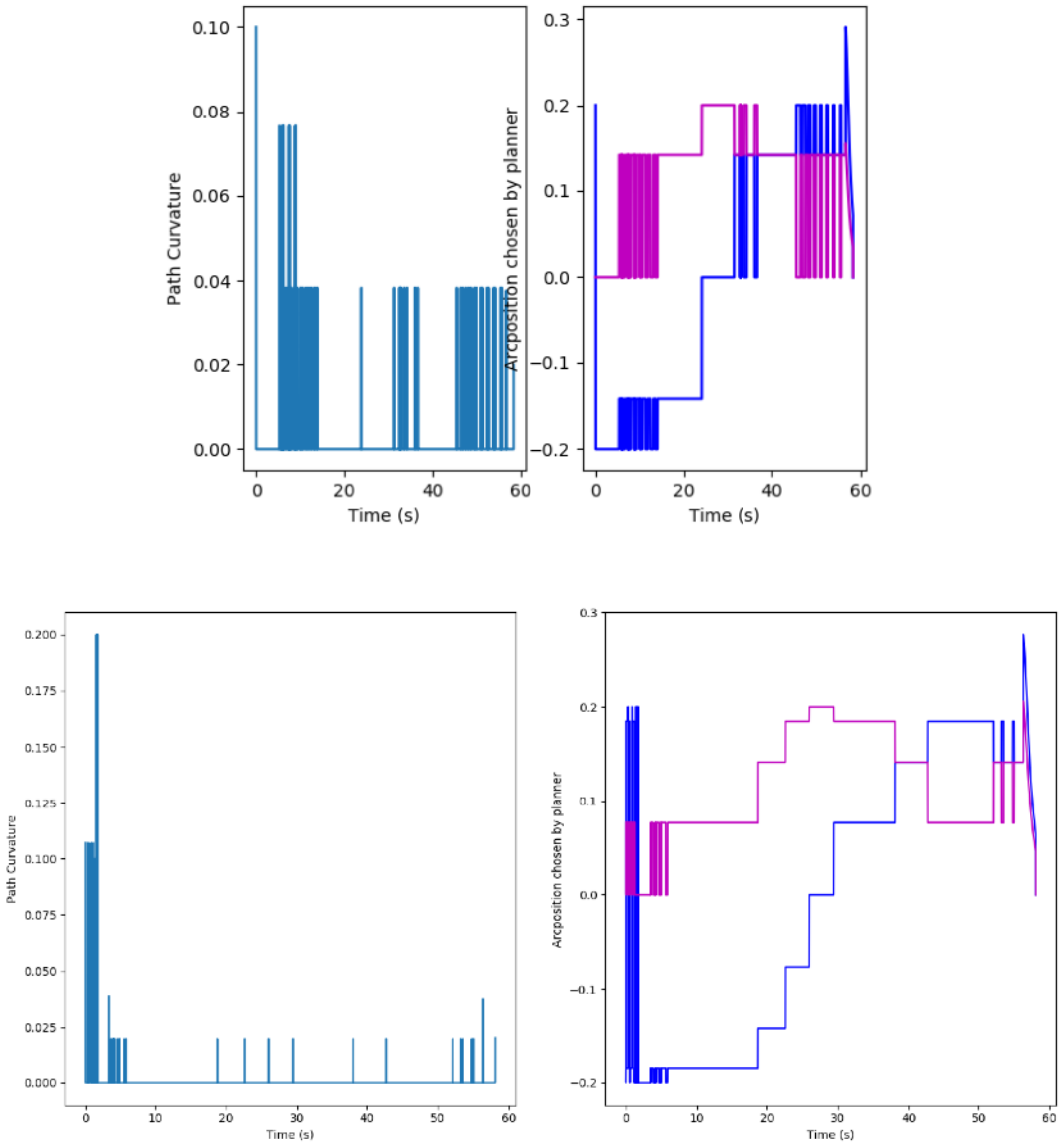


Figures showing the trajectory (blue line) taken by a quadcopter using the Multipoint APF method to avoid an obstacle (depicted by a black square) moving towards it at 0.5ms^-1. a) The setting A = 100 results in an inability to move away from the goal and so results in a collision, indicated by the termination of the blue line before reaching The goal at (0, 5). b) The setting A = 10 sufficiently directs the drone away from and around the obstacle.

the drone can move in, resulting in significant discontinuities in the planner's chosen path. To alleviate this, it was determined that two additional arcpoints would be generated at each timestep. These points would subtend an angle of $\frac{\pi}{8}$ from the previous arcpoint chosen by the planner (Figure …) thus allowing for more options for the planner to consider when altering directions. Specifically, this means that the planner need not change direction by a full $\frac{\pi}{4}$ rad but rather can adjust by $\frac{\pi}{8}$ allowing for a smoother designed path. It can be seen in Figure … that this did allow for a smoother path to be designed by the planner. Table … compares the mean curvature of the resulting path as well as the time taken for the drone to complete the path. It can be seen that the smoother path presents an advantage in the time taken to complete the avoidance manoeuvre. The advantages of smooth paths are discussed in Section … Furthermore, smoother planned paths improve the performance of the PID controller. Large changes in the setpoint can cause the integral term to accumulate over time - an issue commonly referred to as Integral Windup [Control Slides]. Setpoint changes also result in a concern known as Derivative Kick [Matt's webpage] where the rapid change in set point results in a large error derivative. Whilst this is less of a concern with low radius values and there are known methods to alleviate this problem, a smoother path goes further to ensure that the drone can be accurately driven to the goal location. Control issues and their solutions are discussed in Section ….

	Distance of Closest Approach (m)	**Mean Curvature**	**Path Completion Time (s)**
No additional arcpoints	0.23	0.0006	58.29
With additional arcpoints	0.22	0.0005	58.07

Table comparing the use of the Multipoint APF method with and without additional arc points in the case I where the drone must navigate around a line of static obstacles. It can be seen that the addition of two arc points presents a modest improvement on all counts. The method by which mean curvature is determined is described in Section (Evaluation)



Figures depicting the curvature of the trajectories chosen by the planner both without additional arcpoints (Top) and with the addition of arc points as shown in Figure … (Bottom). The additional arc points serve to increase the smoothness of the path as shown by the decreased curvature at given points (Left subplot) and the decreased change in arc position at given times (Right subplot). Table … verifies these findings and discusses the positive impact on the path completion time. REDO THIS

Addition of Collision Cone (CC) to the Multipoint APF method

Additional arc points results in only a modest improvement in all metrics. To further improve this result, the potential field method was reinforced with a collision cone method. This was a development made by Kim et al [APFCC] who showed that the fusion of the collision cone method - another popular method for dynamic obstacle avoidance - with the APF method was reliably able to avoid local minima situations and generate a smooth path.

COLLISION CONES AND THE VELOCITY OBSTACLE (VO) METHOD

The collision cone method, developed by Chakravarthy and Ghose [Collision Cone OG], is a geometric approach to collision avoidance and was extended to the velocity obstacle method by Fiorini and Shiller [VO OG]. Here, a cone is considered which extends from the robot towards

each obstacle. This collision cone defines the set of relative velocities between the robot and the obstacle which would result in a collision (Figure …). It is formalised as

$$CC = \{v_{ro} : B \cap \lambda(v_{ro}) \neq \varnothing\}$$

Where $\lambda(v_{ro})$ is a line extending from the centre of the robot with a gradient of v_{ro} - the relative velocity between the robot and the obstacle - and B is a circular region surrounding the obstacle which has a radius equal to the radius of the obstacle added to the radius of the robot (Figure …) (Not to be confused with the radius of the circle of arcpoints). These are formally given by Alsaab and Bicker as [Improved CC]

$$\begin{aligned} v_{ro} &= v_r - v_o \\ \lambda\left(v_{ro}\right) &= \left\{c_r + \mu v_{ro}\right\} \\ \hat{O} &= \left\{c_o + r : r = r_r + r_o\right\} \end{aligned}$$

Here, v_r and v_o are the velocities of the robot and obstacle respectively, c_r and c_o are the positions of the robot and obstacle and r_r, r_o are the radii of the robot and obstacles respectively. These radii were found through the bounding box dimensions of the robot and obstacles. The cone is therefore bounded between the tangent lines λ_L, λ_R shown in Figure …. Any value of v_{ro} which lies outside of this space will result in a collision. This can be determined by considering the angles from the x-axis subtended by each of these lines as shown in Figure …. [Improved CC]. Alsaab and Bicker show that these can be calculated through

$$d = \sqrt{\left(x_o - x_r\right)^2 + \left(y_o - y_r\right)^2}$$

$$L = \sqrt{d^2 - r^2}$$

$$\phi = \arctan\left(\frac{y_o - y_r}{x_o - x_r}\right)$$

$$\phi_L = \arctan\left(\frac{r}{L}\right) + \phi$$

$$\phi_R = -\arctan\left(\frac{r}{L}\right) + \phi$$

Therefore, v_{ro} lies in the collision cone if it satisfies

$$\phi_R \leq tan^{-1}\left(\frac{v_{ro,y}}{v_{ro,x}}\right) \leq \phi_L$$

In this case, the relative velocity of the robot towards an arcpoint ‘i’ against the obstacle was defined as

$$v_{ro,i} = \left(\frac{d_{r,i}}{R}||v_r||\right) - v_o$$

where R is the radius of the circle of arcpoints around the robot (Section Multipoint UAV Method) and v_r is the current velocity of the robot. This method can easily be extended to accommodate multiple obstacles by considering an individual collision cone to each of the obstacles and ensuring that the relative velocity against each obstacle does not lie within the corresponding cone. Kim et al [APFCC] fuse this method with the potential field method through an additive force if the relative velocity lies within the collision cone. This addition has a constant magnitude μ. To

integrate this into the Multipoint method, a value of μ was added to all arcpoints where the relative velocity lay within the collision cone. μ therefore becomes another tuneable parameter.

The fusion of the collision cone method was tested against the original Multipoint method in the same scenarios as before. As a reminder, these are

I. Obstacle lying directly in the path between the robot and the goal

II. Obstacle moving towards the robot

In the former case, multiple objects were placed next to one another since this was the case in which the original method with $A = 100$ was not able to design a suitable path. Figures … show a comparison of the methods. When using the method without a collision cone, the setting A = 10 was chosen since it has already been determined to achieve success in both scenarios. However, the fused Multipoint APF-CC method uses A = 50 and μ = 40. This value of μ was chosen arbitrarily though would need to be tuned according to the situation in question. A comparison of the methods is shown in Tables … It was found that the fusion of the collision cone method with the Multipoint method was consistently able to complete the path in a shorter time than the original method. Whilst on first glance it seems that the collision cone simply presents another tuneable repulsive potential, this potential specifically acts according to the relative velocity between the robot and the obstacle and is constant. This presents the advantage of taking effect much sooner than the original repulsive term. Furthermore, the additional repulsive term crucially allows the value of A to be increased which, as previously shown, ensures that the drone does not move further than necessary away from the goal. Since the collision cone term incorporates directionality rather than just distance, it will disengage once the robot has moved sufficiently past the obstacle and so can redirect towards the goal sooner. These factors result in a more robust and smooth avoidance solution.

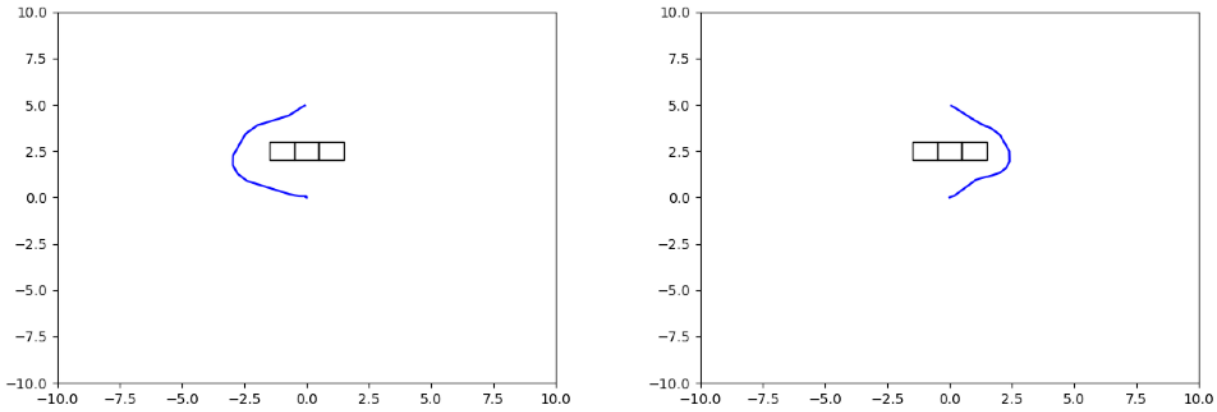
Line of Obstacles	**Distance of Closest Approach (m)**	**Mean Curvature**	**Path Completion Time (s)**
Multipoint APF	0.23	0.0005	58.07
Multipoint APF-CC	0.22	0.0001	49.81

Table comparing the use of the Multipoint APF and Multipoint APF-CC method where the drone must navigate around a line of obstacles directly in its path. It can be seen that the addition of the collision cone presents drastic improvements on all counts, especially the completion time. This is owing to the ability to choose higher A values which aids in driving towards the goal.

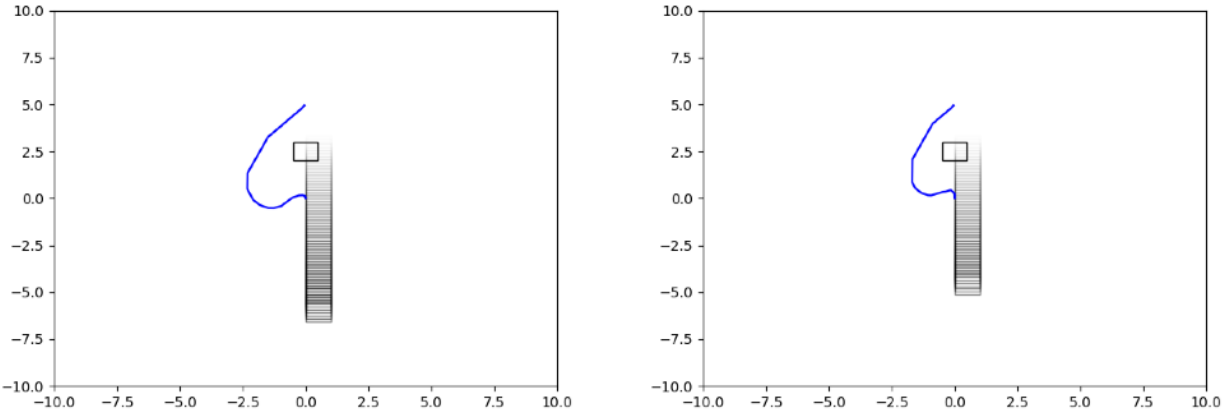
Obstacle Moving Towards Drone	**Distance of Closest Approach (m)**	**Mean Curvature**	**Path Completion Time (s)**
Multipoint APF	0.22	0.0003	57.86
Multipoint APF-CC	0.12	0.0002	48.27

Table comparing the use of the Multipoint APF and Multipoint APF-CC method where the drone must navigate around an obstacle moving towards it at 0.5ms^-1. It can be seen that the addition of the collision cone presents drastic improvements on all counts, especially the completion time. This is owing to the ability to choose higher A values which aids in driving towards the goal.

Whilst the collision cone method proved advantageous in local minima scenarios, a number of scenarios were found where the collision cone addition actually proved to be a disadvantage. One such case which is of particular note is when the obstacle lies directly behind the goal as in Figure



a) Using the original Multipoint APF method, the setting A = 10 is able to direct the drone around a line of static obstacles before leading to the goal at (0, 5). b) The setting A = 50 with μ = 40 using the fused Multipoint APF-CC method is also able to direct the drone around the obstacle but is able to do so with a lower deviation from the obstacle and, as shown in Table …, presents further improvements in curvature and completion time

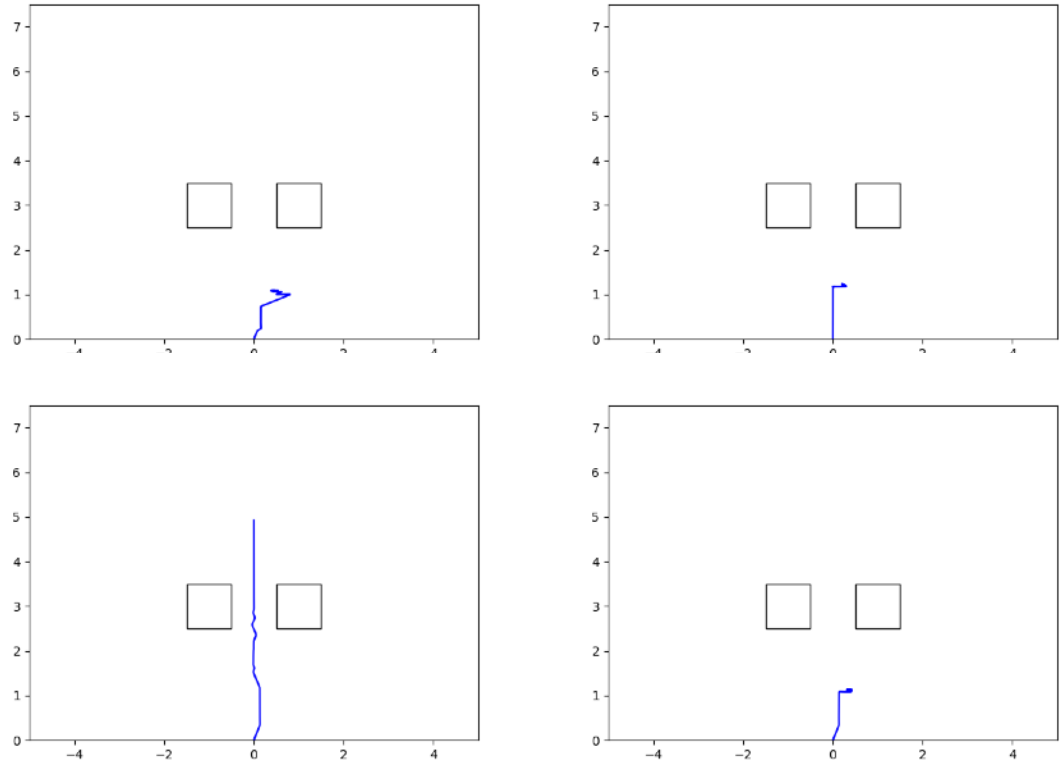


a) The setting A = 10 in the Multipoint-APF method sufficiently directs the drone away from and around the obstacle before leading to the goal at (0, 5). b) The setting A = 50 with μ = 40 using the fused Multipoint APF-CC method is also able to direct the drone around the obstacle but, as shown in Table …, presents further improvements in curvature and completion time

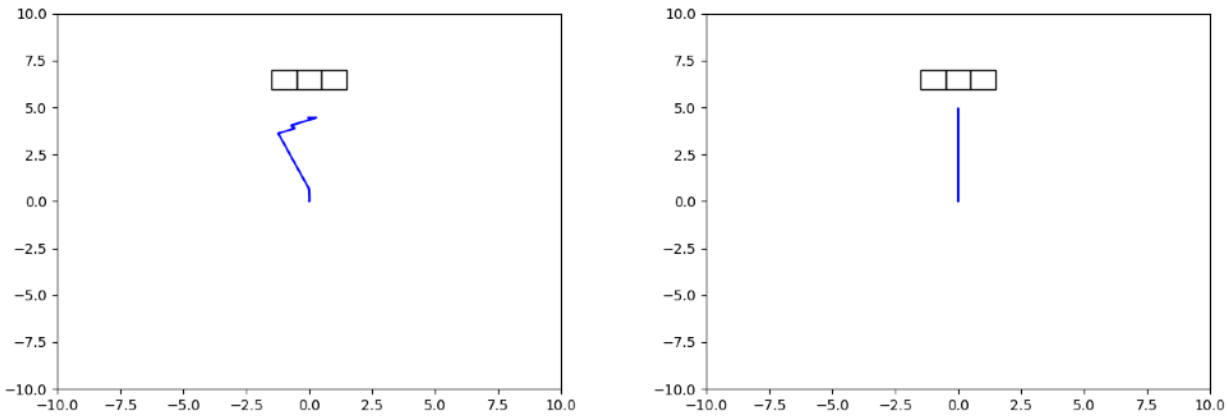
…. In this case, the robot should simply move directly to the goal. However, the robot instead follows a curved path and oscillates near the goal without being able to reach it. This situation illustrates the drawback of the additional repulsive term which is independent of the relative distances of the obstacles and the goal. This was found be alleviated by decreasing the relative strength of the collision cone term with respect to the repulsive term. As such, μ was formulated as

$$\mu = \eta \times C$$

C could now be tuned so that the strength of the collision cone term could be adjusted relative to the repulsive force. In order to realise the full benefits of the APFCC method, the values of A and C must now be tuned according to the required environment. With the optimal values of both



Figures showing the trajectories taken by a quadcopter using the Multipoint APF-CC method to navigate through a gap of 1m between two obstacles. a) A = 50, C = 40. The drone oscillates at the entrance to the gap, unable to move through or around the obstacles. b) A = 50, C = 0. This is found to be a concern with both collision cone and APF methods. c) A = 100, C = 40. Increasing the importance of the attractive term ensures that the planner stays on course towards the goal. As shown in d., increasing C results in stronger oscillations whereas decreasing C or increasing A would result in a straight path to the goal. d) A = 100, C = 80. Increasing the contribution of the collision cone results in similar oscillations as in a. and b.



a) The setting A = 50, μ = 40 is not able to direct the robot towards the goal. It instead deviates from the intuitive straight path and oscillates when approaching the target. This is due to the fact that the collision cone term is independent of distance from the goal or the obstacle. b) A = 100, C = 1. By reducing the relative importance of the collision cone term with respect to the repulsive term.

parameters the collision cone term can deliver on its ability to deliver smoother paths. However, once the drone is close to the obstacle, the repulsive term takes a higher relative contribution to the choice of arcpoint. Figures … for instance show that the planner is able to direct the drone to the goal using the setting A = 100, C = 1. This opens the possibility for these parameters to be tuned online. However due to time constraints, this is presently beyond the scope of this thesis.

The additive terms brings another problem due to the size of the collision cone itself. In narrow passages the method will often times find that all velocities directed towards the goal lie within a collision cone and so no safe paths can be found. An example of this situation is shown in Figure …. As found by [VFF], this is also a problem associated with the APF method; the passage between the obstacles is often disregarded and a path around them is taken as the local minimum. This highlights a significant disadvantage of the Multipoint method as compared to the original potential field method. The latter is able to choose a path as the local minima and traverse it leading to the goal - even though it is not the optimal path. On the other hand, the Multipoint method takes a simplistic approach and only considers the environment at the arcpoints. This leads it to oscillate, almost in an indecisive manner, as it first chooses to move away from the obstacles before returning to its previous arcpoint which was closer to the goal and so on. In this manner, it is not neither to move through the gap or around the obstacles.

As always, the APF portion of the problem may be alleviated by increasing the value of A, encouraging the planner to stay on course towards the goal. The concerns associated with the collision cone term can similarly be alleviated by decreasing the relative strength of the collision cone term (i.e. decreasing C) in these situations. However, Alsaab and Bicker [ImprovedCC] present an improved velocity obstacle method aimed at solving this problem. Here, a laser is used as the sensing mechanism. Each point on the obstacle which the laser detects is expanded to a circle with a radius equal to the radius of the robot. The tangent angles are determined for each of the grown circles and the maximum left tangent and minimum right tangent are determined as λ_L and λ_R respectively. This results in a tighter fit around the obstacle than would be determined by the original collision cone method (Figure …). Furthermore, this allows for the method to work with obstacles of arbitrary shape rather than working with the assumption that all obstacles are circular with known radii. Since the implementation of sensing modalities is beyond the scope of this project, this extension is not considered. However, it is likely to present significant advantages in confined environments.

A final point to be made regarding the collision cone method is that it allows the shape of the obstacle to be taken into account. This is especially of importance in three dimensions where a sphere is not necessarily an accurate representation of the shape of the obstacle [3dCC]. Chakravarthy and Ghose [3DCC] suggest that, to alleviate this problem, the obstacles should be approximated by quadric surfaces which approximate the obstacle. Quadrics present the advantage that when projected to a plane, they are described as conics [PrinceBook] which are straightforward to model. However, it requires a priori knowledge of the shape of obstacles, so that an appropriate quadric can be chosen. Lin et al. [FastUAS] recognise this limitation and so present a method which instead calculates planar collision cones in the horizontal and vertical planes as shown in Figure …. These two collision cones can then be used in conjunction as an approximation to the 3D collision cone. It was with this method that the planar Multipoint APF-CC method was extended to a three dimensional problem.

Extension of the Multipoint APF-CC method to three-dimension

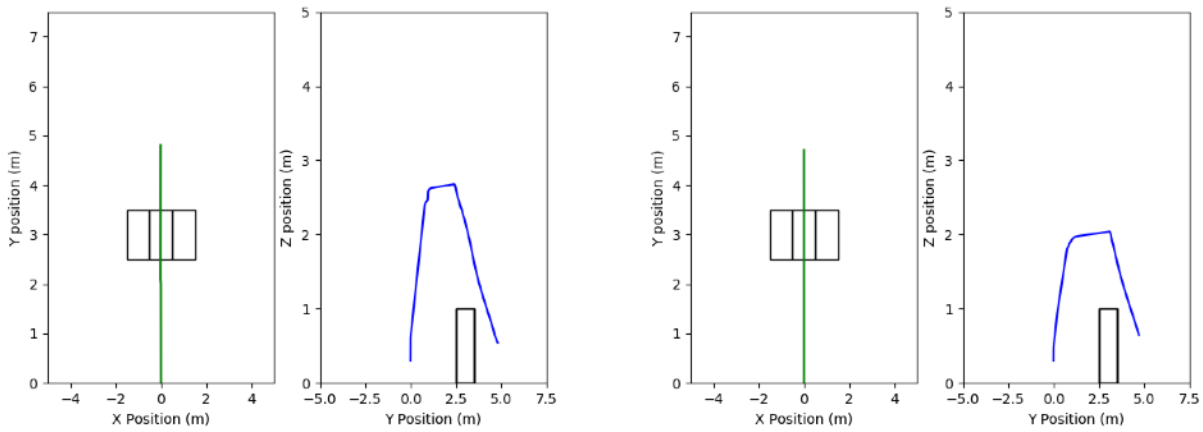
The ability for the planner to consider vertical motion crucially allows it to be deployed in dense environments since the quadcopter now has the capability to plan paths underneath or above objects.

To transfer the Multipoint APF method to three dimensions was a straightforward task as it required that the circle simply be adapted to a sphere of the same radius. Arc points can then be evenly distributed on this sphere as shown in Figure …. This results in 64 arc points as compared to the planar case which required only 8. 4 extra arc points would be added subtending an angle of $\frac{\pi}{8}$ rad from the previously chosen arc point (Figure …) so as to reduce the curvature of the path as was done in the planar case.

Extending the collision cone method to three dimensions was also an accessible task since, as discussed, it simply requires repeating the method as in the planar case to consider the vertical plane as well. Since the dimensions of the robot and the obstacle likely vary in the horizontal and vertical planes (Figure …) it is required that completely separate and independent collision cones

be formulated. The horizontal cone was determined in the x-y plane whilst the vertical cone was determined in the y-z plane if the robot's velocity was higher in the y-axis or the x-z plane otherwise. Note that if an arcpoint's relative velocity lay in both the horizontal and vertical collision cones, μ would be added twice, significantly discouraging the arc point from being chosen as the minimum. As seen in Figure … this requires that C be decreased to accommodate the additional cone. Note also that the ground is considered in the calculation of U_{rep} but is not in the collision cone addition since the ground was assumed to be boundless. This lowers the total repulsive potential from the ground, crucially allowing the planner to choose paths underneath objects rather than continuously searching for paths over them.

Table … considers the same problem as discussed in Section …. That is, navigating the drone around a line of obstacles lying directly in its path. In this case, the drone is able to navigate around the obstacle and achieve its goal at (0, 5, 0.5). It can be seen from this case that the



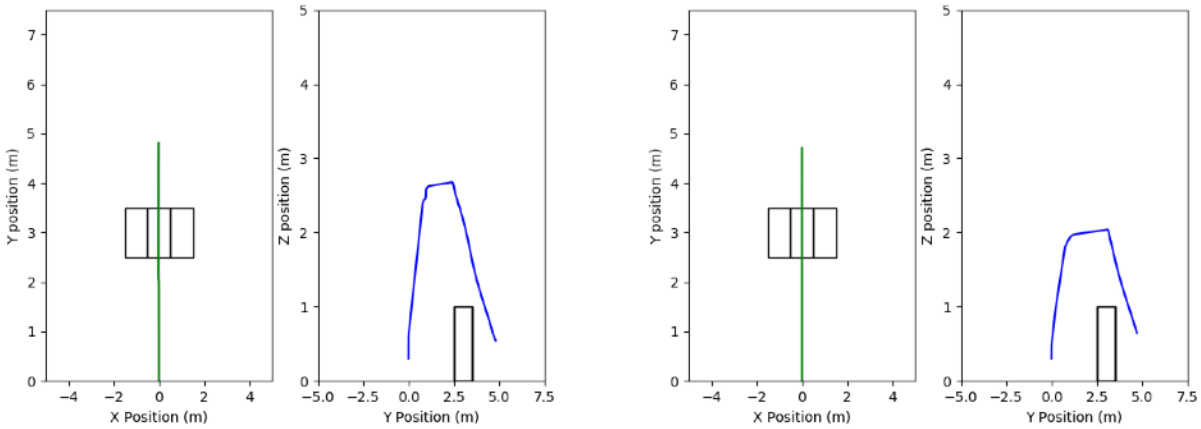
Figures showing the ability of the 3D Multipoint APF-CC method to navigate over a line of obstacles rather than around to achieve its goal at (0, 5, 0.5). a) A = 50, C = 40. The planner considers a large repulsive force from the ground due to its proximity as well as a doubled effect of the collision cone terms resulting in a higher than necessary curve over the obstacles. b) A = 100, C = 10. Decreasing C negates the doubled effect of the CC terms

extension to three dimensions presents significant advantages for the planner's ability to choose shorter, smoother paths.

Line of Obstacles	**Distance of Closest Approach (m)**	**Mean Curvature**	**Path Completion Time (s)**
Planar Multipoint APF-CC	0.23	0.0005	58.07
3D Multipoint APF-CC	0.23	0.00008	38.3

Table comparing the use of the planar and 3D methods when attempting to navigate around a line of obstacles. As the 3D method allows the planner to navigate over the obstacle rather than around it, the trajectory becomes significantly smoother and shorter presenting a significant improvement in the path completion time.

As is seen in Figure … when attempting to traverse underneath an obstacle, the drone often is not able to begin an avoidance manoeuvre since the repulsive force from the ground is equivalent to that of any other obstacle. As such, the planner is not able to consider the fact that it can move closer to the ground. To alleviate this, a new term is introduced as $G_{discount}$. This is a discount



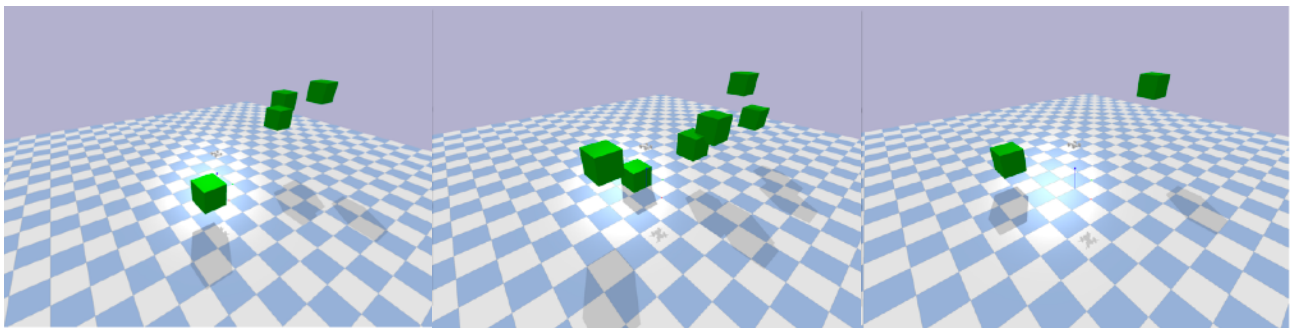
Figures showing the ability of the 3D Multipoint APF-CC method to navigate over a line of obstacles rather than around to achieve its goal at (0, 5, 0.5). a) A = 50, C = 40. The planner considers a large repulsive force from the ground due to its proximity as well as a doubled effect of the collision cone terms resulting in a higher than necessary curve over the obstacles. b) A = 100, C = 10. Decreasing C negates the doubled effect of the CC terms

applied to the ground to decrease the contribution to U_{rep}. By applying varying values of $G_{discount}$ to the problem shown in Figure …, a functional setting for $G_{discount}$ was found as 1.5. However, with the new discount, it is important that A be increased to compensate since, as shown in Figure …, without this increase, when far from the goal, the discounted repulsive term finds directions towards the ground as the new minimum and so does not prioritise movement towards the goal. Once a stronger tuning of A and C were applied (A = 800, C = 2), the planner was able to reach its goal.

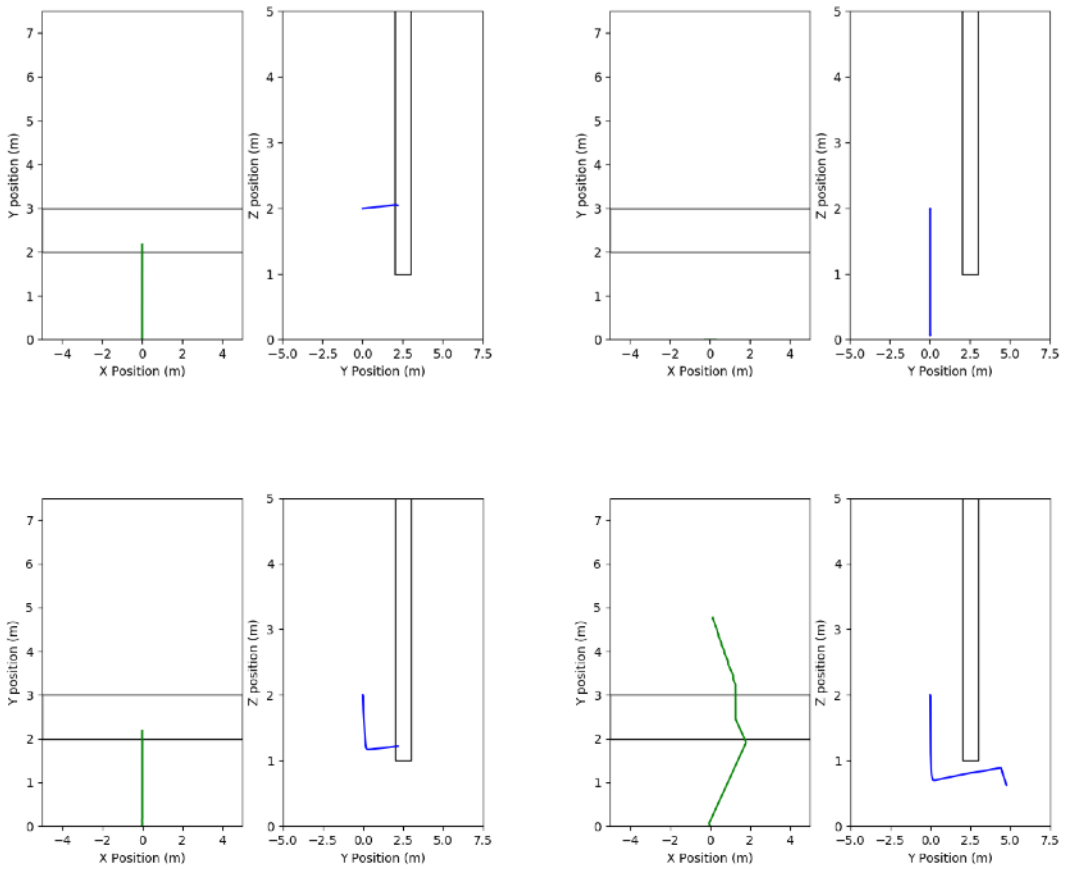
The discount term can be applied either by reducing the value of η applied to the ground or, as was done in this case, by multiplying the distance '$d_{i,obs}$' from all arcpoints to the goal by $G_{discount}$, so that the planner perceives the ground to be further than it truly is.

Evaluation of the Multipoint APF-CC method

A series of tests were designed for the Multipoint APF-CC method (and are used for the GNN method described in Section …) to determine the performance of the method in a number of scenarios. These tests consider the drone's ability to navigate through dense environments containing both static and dynamic obstacles. The environment was generated randomly so that the ability of the method to generalise can be considered. An example of such an environment is shown in Figure … These require the planner to navigate around, above and under obstacles to achieve the goal. **TESTS RUNNING NOW**



Figures showing randomly generated environments used to test the performance of the Multipoint APF-CC method



Figures showing the ability of the 3D Multipoint APF-CC method to navigate under a tall obstacle attempting to reach a goal of (0, 5, 0.5) from a starting position of (0, 0, 2). a. (A = 1e3, C = 20) no discount provided for ground. The planner is therefore not able to direct towards the ground. b. (A = 100, C = 20, $G_{discount}$ = 1.5). Without a compensated increase in A, the planner is not able to move towards the target since moving towards the ground becomes the minimum. c. (A = 1e3, C = 20, $G_{discount}$ = 1.5). A similar problem as in Figure … occurs where the planner is not able to navigate through a narrow gap. d. (A = 8e2, C = 2, $G_{discount}$ = 1.5) As before, reducing the collision cone term allows the planner to navigate safely under the obstacle and achieve its target.

Genetic Neural Network Method

In recent years, significant advances have been made in the fields of deep learning and reinforcement learning. These techniques have had a significant impact in all forms of machine learning but have a particular bearing to the problem of obstacle avoidance and navigation in mobile robots. As such, they merit further examination.

Deep learning is defined by LeCun et al [LeCun Deep] as a method by which multiple levels of abstraction regarding the representation of data may be learnt. This is accomplished by processing the data through multiple layers. These layers are built of nodes and are connected (via edges) to nodes of neighbouring layers. When fully assembled, the stack of layers is referred to as a neural network (Figure ...) with each edge being associated with a weight 'W' and bias 'b', and each layer associated with some 'activation function' a(x). Data may then be input into the network. As each data point traverses along an edge, it is transformed according to the function

$$e_j(x_i) = W_j x_i + b_j$$

Where j refers to the edge and i refers to the data point. After passing along the edge, the data point arrives at the node where it undergoes another transformation - namely the activation function. Common choices for the activation function are: ReLU (rectified linear unit), sigmoid and tanh, though these are often chosen empirically. This procedure continues forward through the network until the output layer which presents the final result. See [GoodFellow] for further details. A widely known example of this is of the MNIST dataset [Tensorflow Keras Tut], where a series of images containing handwritten digits is passed through a network. Here, the input is the raw pixel data of the image whilst the output is a vector of 10 probability values, each pertaining to the likelihood that the image was of the associated digit.

Once an output has been reached, it is passed through a predefined loss function. In the case of the MNIST dataset, this may be the difference between the output and a vector of zeros containing a value of 1 in the cell associated with the 'true' digit. The aim of training a network is to choose the parameters (the set of W_j and b_j) which minimises this loss function. This is typically achieved by choosing from a number of non-linear optimisers [Data Science Optimisers]. Following this, another batch of data (perhaps another image) is provided and the process is repeated. In theory, after presenting the network with a significant amount of data, the weights and biases should be set so that the output predicted by the network almost exactly matches that which is expected.

Reactive Collision Avoidance using Evolutionary Neural Networks

In [ReactiveCollision], Eraqi et al propose a method by which a neural network can be used towards the problem of collision avoidance. The network takes as its input the data provided by a rangefinder and outputs a pair of steering forces which may be used to determine a steering wheel angle. This illustrates an advantage that network based methods have over both traditional methods (Section …) and also reinforcement or DRL methods (Section …); the network is able to take on much of the load from the controller if an appropriate output is defined. Section … goes on to explain how this advantage applies to a quadcopter.

Eraqi et al. then choose a genetic algorithm as the optimiser for the neural network. Genetic algorithms fall under the category of evolutionary algorithms - a branch of optimisation which is inspired from nature [EAReview]. Genetic algorithms specifically is inspired by the Darwinian conception of natural selection. The process is described in Algorithm …

1. Initialise a random set (population) of solutions
2. Evaluate each solution (member) in the set against a fitness function
3. Choose the members in the population with the highest fitness as parents
   For each pair of parents

Generate new solutions (children) by crossing over part of the solution from one parent with part of the solution from the other parent

Do this until the new set of solutions is the same size as the initial solution

4. For each member of the new population

Randomly mutate the solution

Through this seemingly random process, the algorithm is able to optimise over a large search space. This is an important advantage for use in neural networks where each member is defined by the list of weights and biases which define the network (Section …). This is also useful for loss functions where gradient information is not available. This is especially the case with the task of obstacle avoidance. Here, we may define a loss function such as the lifetime of the robot before collision (as was done in [ReactiveCollision]). This loss is non differentiable and so a gradient based optimiser cannot be used.

Deformable Virtual Zone

In [FuzzyDVZ], Baklouti et al. apply a controller based on a deformable virtual zone (DVZ) as proposed in [OGDVZ] to the task of obstacle avoidance in a mobile robot. The authors demonstrate it's effectiveness in a three-dimensional situation as the robot is able to pass underneath obstacles if there is enough room.

The DVZ is a space around the mobile robot formalised as an ellipsoid.

$$\frac{(x-a_x)^2}{c_x^2}+\frac{\left(y-a_y\right)^2}{c_y^2}+\frac{(z-a_z)^2}{c_z^2}=1$$

The values of $\{c_i, a_i\}_x^z$ are set according to the velocity of the robot. ( ) shows the particular case when the drone is primarily moving in the x direction. However, this can be trivially rearranged to instead take the case where the robot is primarily moving in the y or z directions

$$c_x = \lambda V^2 + c_{min}$$

$$c_y = c_z = \frac{\sqrt{5}}{3}c_x$$

$$a_x = -\frac{2}{3}c_x$$

$$a_y = a_z = 0$$

Where lambda and c_min are parameters used to control the size of the DVZ. With this formulation, derived from [FuzzyDVZ], the DVZ forms an ellipsoid which lies slightly infront (front defined by the current direction of motion) of the robot and is elongated along the axis in which the robot is primarily travelling. In the body frame of the robot, a point on the DVZ can be defined as

$$x' = d_h(\theta,\phi)\cos\theta\cos\phi$$
$$y' = d_h(\theta,\phi)\sin\theta\cos\phi$$
$$z' = d_h(\theta,\phi)\sin\phi$$

Where d_h is the distance to a point along the ellipsoid's surface and theta, phi are angles defined in the spherical coordinate system (Figure …). Here, prime notation indicates that coordinates are with reference to the body frame of the robot. To determine $d_h(\theta,\phi)$, ( ) must first be rotated to align with the world frame.

$$\overrightarrow{x} = R\overrightarrow{x'}$$

where R is the rotation matrix corresponding to the orientation of the body frame with respect to the world frame. Then, ( ) may be substituted into ( ) to yield an expression for D_h (Appendix D). The DVZ is deformed when an obstacle enters into the zone and is formalised as

$$d(\theta,\phi) = \begin{cases} c(\theta,\phi) & \text{if } c(\theta,\phi) < d_h(\theta,\phi) \\ d_h(\theta,\phi) & \text{otherwise} \end{cases}$$

Where $c(\theta,\phi)$ is the distance from the robot to the obstacle. Note that this is only meaningful if the object is inside the DVZ. With this definition, the DVZ allows for a representation of the environment which incorporates the shape of the obstacle without explicitly requiring any prior knowledge; the only explicit information comes directly from the rangefinder data. Furthermore, the information is dependent on the velocity of the robot itself. This dependence on the robot state allows for a more meaningful representation of the environment than would be found if simply using rangefinder data as is done in [ReactiveCollision].

The deformation of the DVZ may be quantified by an 'Intrusion' term I. As noted by Baklouti et al, the choice of Intrusion is critical to the robot's reactivity. As such, the intrusion term introduced in [FuzzyDVZ] is used

$$I = \int_{\theta=0}^{2\pi} \int_{\phi=0}^{\pi} \frac{d_h(\theta,\phi) - d(\theta,\phi)}{d(\theta,\phi)} d\phi d\theta$$

This intrusion term is a function not only of the environment but, by the choice of $\{c_i, a_i\}_x^z$ is also dependent on the robot's state. It should also be noted that, due to the denominator in ( ), I can grow asymptotically as the distance between the robot and obstacle decreases. As such, a severe penalty is placed on moving towards obstacles.

Network Design

The network used draws from that proposed by Lei et al [CNNLidar] and is shown in Figure …. The network takes as input the list of $\{d(\theta,\phi)\}_{\theta,\phi=0}^{2\pi,\pi}$ defining the DVZ. As described in Section …, a number of authors have found that incorporating memory into the network has shown to increase the ability of the network to represent the motion of both the robot and the environment. To add this feature, a solution inspired by [DeepMindAtari] was applied where a history of DVZs from four timesteps are concatenated to form the input. $\{d(\theta,\phi)\}_{\theta,\phi=0}^{2\pi,\pi}$ is 64800 (360 x 180) long meaning the complete input vector has (4 x 64800 =) 259200 elements. This is considerably larger than the 360 input elements used by Eraqi et al. This results in significantly more parameters (weights and biases) required to define the network, thereby increasing the total search space of the optimiser.

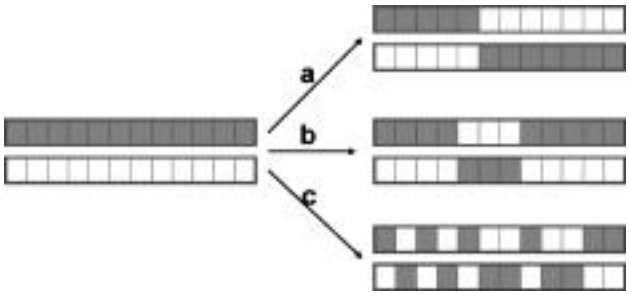
Lei et al [CNNLidar] address this through the use of a convolutional network [Goodfellow]. Convolutional layers are designed to effectively encode the spatial relationships and characteristics within an input through the use of filters. This has the combined effect of producing a meaningful representation of the environment and reducing the total network parameters. The following paragraph gives the full architecture of the network

Architecture:
SUBJECT TO CHANGE

The three outputs of the network are: roll angular velocity, pitch angular velocity, vertical velocity. These are all defined in the body frame of the quadcopter. As mentioned, this presents an advantage for the PID controller [Appendix …]. This is since a controller is no longer required on the x-y position. Instead, the roll-pitch angles can be determined directly.

GA Composition

The genetic algorithm is defined with a population of 2000 members. Each member is defined by the set of neural network weights and biases as shown in Figure … Therefore, one may consider each member of the population to be an individual neural network with defined weights and biases. In each iteration, the 400 members with the highest evaluated fitness (defined below) are chosen to produce the next generation of members. As in Algorithm …, reproduction occurs through a 'crossover' process where part of the solution from one parent is combined with another part of the solution of another parent. In particular, a single-crossover scheme is used. Here, a point in the solution (i.e. a particular element in the vector of weights and biases) is chosen as the crossover point. Then, the portion of one member leading up to this 'crossover site' is concatenated with the portion of the second member which follows this point. See Figure … [CrossoverFig] for an example. The single-crossover scheme is shown in [CrossoverCompare] to perform optimally in a number of tasks over other methods.



Crossover scheme with two members. a. Single-crossover b. Two-point crossover c. Uniform crossover [CrossoverFig]

The crossover site was chosen randomly, by sampling a gaussian (Figure …) with

$$\mu = L_s/2$$
$$\sigma = L_s/8$$

where the solution length L_s is the total number of parameters required by the network. Whilst more optimal methods [BetterGA] exist for choosing the crossover site, the ambiguity between the solution parameters and the fitness function largely restrict the optimisation to randomisation. Further work could bring to light methods to improve the convergence of the optimisation,. Note that each generation completely replaces the previous generation so that, in this case, the crossover procedure is repeated ten times for each pair of parents so that a new set of 2000 unique solutions are generated.

The new generation then undergoes a mutation process. Here, each parameter in each solution is randomly - with a defined 'mutation probability' - changed to another. The mutation is done by adding a value sampled from a gaussian of mean 0 and standard deviation 0.25. Rocha et al [AIProgress] found that this gaussian mutation was critical to ensuring that the genetic algorithm optimisation showed performance comparable to that of gradient based optimisers.

Each member is evaluated against a fitness function. In this case, this is defined as

$$f = -\int_0^T te^2(t)dt - \sum_{t=0}^{T} I_t$$

Where $e(t)$ is the distance from the robot to the goal at time t

$$e(t) = ||\overrightarrow{x_g} + \overrightarrow{x_r}(t)||$$

And I_t is the DVZ intrusion at time t. T is the total time elapsed in simulation which is capped at 120s. The first term in ( ) is drawn from [PSOPID] and aims to encourage the robot to move towards the goal whilst the DVZ intrusion term encourages obstacle avoidance. Note that the fitness is assigned after the simulation is complete and so learning does not occur online. The aim, therefore, is to find a set of parameters which may generalise to different environments.

TRAINING NOW

Comparison of Methods

Evaluation Metrics

This section provides a brief description regarding the metrics used for the evaluating the performance of each system. Where appropriate, an explanation on how they are determined and their rationale is discussed.

- **Computational Time**

The time taken for the planner to perform its task at each timestep. The path's computational time is given as the mean computational times over all t. This should be minimised to allow for reactive behaviour and strong real-time performance.

- **Distance of Closest Approach:**

The shortest distance from the robot to any given obstacle. This metric aids in understanding the optimality of the path chosen. Shorter distances are considered to be more optimal as it ensures that the drone is able to reach its goal in the shortest amount of time possible since it does not stray further than necessary from the original straight line path.

- **Path Curvature**

Similar to the previous metric, this is a consideration of its optimality. Lower curvatures are indicative of smoother paths and so can be traversed in a shorter time. This is since the quadcopter is not required to accelerate or decelerate rapidly at any given time. Furthermore, smoother paths present an advantage in terms of the power consumption by the quadcopter rotors. This is due to the fact that the rotors are not required to rapidly increase or decrease their angular velocities. It can be seen in Figure … that smoother paths result in smoother rotor angular accelerations. Curvature at time t is calculated as in ( ) [NiloySlides] with the path curvature defined as the mean curvature over all t.

$$C_t = \frac{1}{2} || \left(\frac{1}{2} \overrightarrow{x_{t-1}} - \overrightarrow{x_t} + \frac{1}{2} \overrightarrow{x_{t+1}} \right) ||$$

Where $\overrightarrow{x_i}$ is the position of the quadcopter at time i.

- **Rotor Curvature**

The rotor curvature verifies the effectiveness of the previous metric by considering the curvature of the angular velocity squared (Appendix Control) of each rotor. Smaller curvatures are considered to be more optimal since this requires less power consumption by the rotor. This is calculated as

$$\overrightarrow{C_{\gamma,t}} = \frac{1}{2} \left(\frac{1}{2} \overrightarrow{\gamma_{t-1}} - \overrightarrow{\gamma_t} + \frac{1}{2} \overrightarrow{\gamma_{t+1}} \right)$$

Where $\vec{\gamma}$ refers to the vector of four inputs to the rotors (i.e. the angular velocities squared of the rotors). Note that the l2-norm is not used in this case as in ( ) since the curvatures of all rotors are independent. In this case, the rotor curvature of the path is again a four vector denoted as the maximum over all t for each rotor. This is a representation of the largest discontinuity in the rotors' lifetime and should be minimised to conserve power.

- **Lifetime/Pathtime**

If the quadcopter is able to reach the goal, the path-time (time to reach the goal) is used. This should be minimised. If the quadcopter collides into an obstacle, the lifetime is used since the simulation will be reset. The lifetime is the time elapsed before the collision and should be maximised.

- **Pathlength**

Serves as a supplementary to the path-time and closest approach and is simply the total length of the path taken by the quadcopter. This should be minimised. It is considered since, in static environments, the path length is independent of the drone's velocity, therefore, of the performance of the controller (assuming no overshoot occurs). By contrast, the path-time can also be reduced with a more accurate controller. Since the performance of the PID controller (Appendix Control) is beyond the scope of this project, the path length is considered to be a stronger representation of the planner's capability.

Literature Review

This section considers a number of existing machine learning methods of obstacle avoidance and weighs their comparative advantages. It aims to show that learning agents can exhibit powerful behaviour in the field of obstacle avoidance. However, the success of the agent is highly dependent on the ability of the system to generalise to previously unseen environments and to learn a strong representation of the environment for avoidance rather than user input for imitation and is extremely sensitive to reward structures and training methodologies. It is with this in mind that the method described in Section … was chosen. The method allows for a definitive loss function to applied which can generalise to a wide variety of situations and, in fact, was shown by Eraqi et al [ReactiveCollision] to be generalise to more complex situations than it was presented during training. This particular facet was deemed to be the method's advantage over those described below. However, each of the following techniques - particularly deep reinforcement learning (DRL), has shown a strong performance towards solving the obstacle avoidance problem and are worth consideration.

Learning Methods to Obstacle Avoidance

Deep Learning has gained significant attention in the past decade [Deep Learning for Robotics] for its ability to make strong predictions and to generalise to unseen data well. Furthermore, no direct restrictions are placed on the form of the input or output. For these reason, it has a strong applicability to navigation problems and a number of network designs have been proposed with the purpose of avoiding obstacles.

One such design is proposed by Djugash and Hamner [nnObstacleAvoidance]. Here, a human driver is presented with the vehicles display and is instructed to navigate through multiple environments containing static obstacles. Throughout this procedure, the state of the vehicle, consisting of both its linear and angular position and velocity, is recorded at each time step. This serves as the training data which the network is supplied. As inputs, the network is given the distance and angle to the goal as well as the distance to obstacles in discretised segments in front of the robot. The latter point is a valuable addition since network architectures often require that the shape of the input vector remain consistent in all situations; however, it is unlikely that the number of obstacles in every environment remain the same. This is cleverly resolved through the discretisation of the space in front of the robot into equiangular segments. Then, the distance to the nearest obstacle in each segment is taken to be the input. The network then outputs an angular velocity command and the state of the car is driven to match this command. In training, the network aims to match the angular velocity that was produced at that time step when a human was controlling the robot. After only 100 training iterations, the network was able to match the route taken by the human and so was able to drive to the goal position. However, as pointed out by Djugash and Hamner, this system is able only to reproduce steering angles rather than learning to avoid obstacles, as such, there is no real knowledge of the situation presented. Furthermore, the method requires a significant amount of data - as is the case with any machine learning technique. In the case of quadcopter navigation, however, it may not be feasible to collect enough training data (of humans controlling the quadcopter) in a variety of different environments. To little data, however, and the network is likely to overfit to the particular situation shown to it, as was the case with Djugash and Hamner's experiment. A similar concern lies with the architecture proposed by Kim and Chen [nnIndoorNavigation]. This is a similar case, where a deep classifier attempts to mimic the the flight commands of an expert pilot to navigate through indoor environments. Whilst the system was able to navigate through a number of indoor environments with a success rate of up to 80%, when faced with a unique situation (which was not included in the training set), this rate dropped to 60%. Similar issues occur with a sub field of reinforcement learning known as 'Imitation Learning'.

Reinforcement Learning is a well established mode of machine learning which concerns itself with the ability of systems to interact with their environment. A popular example of this is the cart pole problem [Cartpole], where the system must learn to balance a pole on a moving cart. To

accomplish this, reinforcement learning draws on the 'Markov Decision Process' [BRML] in which the aim is to maximise an expected utility (or value) which is provided by the environment. In reinforcement learning, this is formulated as [RL Slides]

$$V(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots$$

where V is the 'value' function, evaluated at a given state s_t , r_t is a reward provided by the environment and gamma (between 0 and 1) is a discount factor to ensure convergence when the problem is temporally unbounded. The agent should then choose actions which result in the total value being maximised. However, these values are unknown by the agent and so, instead, it must maximise the expected result. This is formalised in the Q-value as

$$Q(s_t, a_t) = < V(s_t) >_{s_t \in S}$$

Where S denotes the set of possible states. Now, the agent must choose the action at any given time step, which maximises its Q-value. Since these are initially unknown, the agent must explore the environment to determine the Q-values.

Imitation Learning builds on the framework of reinforcement learning by learning a policy (set of actions) which best mimics an expert pilot's set of actions [DAgger]. This method is referred to as 'behaviour cloning'. However, as before, these require significant amounts of training data and do not necessarily transfer to previously unseen environments. Methods exist to compensate for this loss, such as presented by Ross et al. [DAgger] which attempts to aggregate data from the pilot's behaviour which the agent is likely to encounter but was not previously observed. In doing so, the agent is able to understand how to recover from situations which it has not encountered and, thus, avoid the effect of compounding errors [RL Robotics Survey]. A related concern in all of the above methodologies, is that the performance of the methodologies is highly dependent on the ability of the pilot. In each case, it is assumed that the pilot will always choose optimal actions and, as mentioned, the system simply learns to reproduce this behaviour. Li et al. [OIL] successfully alleviate this problem by considering the advantage or disadvantage of imitating multiple imperfect teachers upon which a learnt policy is built. Whilst the method certainly contains its own flaws, it successfully generalises the behaviour cloning method to encourage the UAV to learn to navigate rather than simply imitate. As the method is applied to more complex tasks, its applicability outside of simulation and in denser environments will become apparent.

Perhaps the most popular form of machine learning solution applied to robotics is that of deep reinforcement learning. This is a recently developed amalgamation of deep learning - which provides a representation of the environment - and reinforcement learning. As succinctly put by Prof. Fridman [Deep RL Lecture], "the promise of deep reinforcement learning is building an agent which uses that representation to achieve success in the world". This technique therefore has tremendous applicability to robotic problems.

The 'deep' component of DRL concerns itself with the Q-values. As previously mentioned, reinforcement learning begins with the premise that the Q-values are not initially known and need to be determined. DRL puts forward a deep network which instead will act as an approximator for the Q-value functions. Once trained, the deep network can then take the state input (through whichever sensing modalities are available) and return the determined Q-values for each action. With these, the reinforcement learning agent can take over and choose to continue exploring the space - providing the network with further data with which to train - or to perform the action with the highest Q-value. This is a subtle yet powerful difference from the vanilla reinforcement learning agent which only has access to a state-action table containing Q-values. This is of especial importance in navigation problems where the state is generally a continuous variable (e.g. distance to obstacle, distance to goal etc). In fact, Huang et al [RLNNProblem] were able to show that the use of a neural network presents marked improvement in an agent's ability to avoid obstacles.

Qiao et al. [nnRLAvoidance] consider this advantage when designing an intelligent behavioural controller for the purpose of obstacle avoidance. The details of behavioural controllers are beyond the scope of this thesis, though [Handbook] and [BehaviourRL] provide thorough explanations on

the topic. In [nnRLAvoidance], the authors use a neural network to produce Q-values for each of the actions their robot can take - to move forward or to turn by varying degrees in either direction. After exploration, the RL agent chooses the action according to a greedy policy (i.e. taking the action with the highest Q-value) at each step. This means that the set of actions can be constrained according to the capabilities of the robot. In fact, Qiao et al. were able to show that this method does allow the planner to direct the robot to the goal in a dense static environment using onboard sensors. However, the robot does not seem to give a significant consideration to optimality;It does not appear to take routes sparse in obstacles or the shortest route but only one in which the robot moves towards the goal and away from nearby obstacles in each time step. This is, in part, due to the manner in which rewards are chosen. The agent receives a constant numerical reward for moving closer to the goal and a negative reward for moving away and similarly for moving towards or away from obstacles.

Similar concerns regarding rewards can be found with the design proposed by Wang et al. [DDPGDriving], which was unable to avoid obstacles since the reward function considered only staying on the provided course, and that proposed by Vitelli and Nayebi [CARMA]. The latter reward structure aimed at maximising the speed of the robot. This inevitably led to unstable behaviour by the agent. Despite this, the authors have designed an incredibly innovative and novel network architecture which combines real time image input with a memory based architecture - the recurrent neural network (RNN) [RNNLiterature]. This allows the agent to consider past information as part of its evaluation of the Q-values leading to a more accurate prediction in dynamic situations. Singla et al [MemoryDeepRL] also consider the use of the recurrent neural network in their application of DRL to UAV obstacle avoidance. The use of the RNN proves advantageous in allowing the agent to consistently increase its ability to avoid any collisions as training commences.

Conclusion + Remarks

Appendix

Appendix A: Control

Appendix B: Parrot AR Drone

Appendix C: Code

Appendix D: DVZ derivation