DD2380 - Assignment 1

GROUP 27

Navin Vincent
2000-12-04
navinv@kth.se

Michal Sitarz
2001-02-22
sitarz@kth.se

15th February 2024

Abstract

We describe a path-planning and execution framework for autonomous vehicles operating in a predefined virtual environment. This work was tested and validated on several virtual terrains where the autonomous agent had to navigate to the goal position. Our approach involved using a variant of the well known A* algorithm, known as Hybrid A*, with an augmented node transition rule that captured the continuous state of the agents in the given grid cells. This was followed by improving the quality of the solution through numeric optimisation to make it more traversible for the agents. Finally, we constructed dynamic controllers to guide the agents reliably through the found path. We demonstrate the versatility and robustness of the controllers on the various terrains and it's capability for high precision vehicle guidance. Thus, we have empirically validated the benefit of such PID based controllers for navigation in autonmous vehicles.

1 Introduction

This report aims to address the problem of path planning and execution by an autonomous car and a drone operating in a known environment. We assume that the robot has sensing and localization abilities and that it has access to a predefined map. The goal was to finish the task by navigating to the target location in a predefined terrain using as little time as possible. This work was carried out in the virtual environment called 'Unity' and the main purpose of this endeavor was to empirically validate certain methods of path planning and execution through such simulated experiments. As such this work is built on already existing work on search algorithms for path planning [4]. There exist significant challenges when you try to develop a path-planning algorithm that is suited for environments that allow free navigation. This stems from the fact that all of the vehicle controls are continuous which in turn leads to continuous trajectories making the optimisation problem quite complex. As Dolgov et al. [4] noted, it becomes difficult to give theoretical guarantees about the achieved optima although in practice it is often in the neighborhood of the global optimum.

Even so, the problem of continuous navigation and path planning for unmanned vehicles is well-studied. Particularly, the DARPA Urban Challenge (DUC), where vehicles had to freely navigate parking lots managed to produce a lot of innovative solutions to the above problem [3, 12, 14]. Once the path has been planned, executing it is also acknowledged to be a difficult problem. This is especially true when the environment has a strong topological structure. In our case it was important for the vehicle to observe the underlying topological structure since each collision halved the allowed acceleration for the vehicle, thus increasing the time it took to get to the goal. As such, we developed fairly sophisticated controllers to navigate sharp curves and U-turns carefully. In particular, for the car, we implemented a dynamic system that reacts to sharp turns and follows the generated path extremely precisely and relatively fast. Whereas, for the drone model we used artificial physics for stable path following. The developed solution adapts some of the ideas from the literature (see Section 3) and we incorporate our ideas to suit the Unity environment and make the agent navigate the terrains successfully.

In the end, we were able to finish the task of reaching the goal in sufficiently good time and were among the top 6 groups in terms of completion time (see Section 4). The controllers we employed resulted in us performing better than most groups in terrains which involved several sharp curves and U-turns.

1.1 Contribution

As mentioned earlier, the problem of autonomous navigation in a continuous free space is well studied and for the most part, we only adapted these known solutions to suit the virtual environment. However, we feel that the path execution in virtual environments using PID controllers with theoretically simple and transparent modifications such as the ones we have presented in this report are not as common. Through this report, we have empirically demonstrated the viability of such controllers in challenging environments. The major contribution of this paper is designing a dynamic controller. Compared to most of the other groups (see Section 4) the car follows the path with very high precision, not straying away from it. This is crucial in the field of autonomous driving as if the car were to miss the trajectory even slightly, it could end in a crash. Furthermore, our controller proved to be extremely flexible, regardless of the terrain, which is a big success, as it does not require fine-tuning for different terrains. On top of that, it can traverse the terrain at a relatively high speed, still staying on the designated path.

1.2 Outline

In the upcoming Section 2 we dive into the literature to outline where we got the inspirations and ideas from. This culminated in developing our final method as it is described in Section 3. There we discuss how we solved the given problem in-depth, and discuss which steps we had to undertake to get the vehicles to complete the course with speed. The outcome of this method was summarized in Section 4. Furthermore, we analyzed and compared our results with other groups, and identified some limitations of our approach. Finally, we conclude the paper in Section 5, where we summarize what we have accomplished, whilst also discussing possible future improvements.

2 Related work

The problem at hand already had a reasonably well-defined map of obstacles that could form the basis of the configuration space. On top of that, the time required for planning was not treated as a limiting factor in our experiments. Thus, we chose a variation of the popular A* algorithm [7] over the RRT* algorithm[8].

During the literature review, we came across papers that made use of variations of the A* algorithm to generate paths that were more 'drivable' for cars following the Kinematic car model. Particularly, a very influential paper by Dolgov and Thrun [4] describes a variant called hybrid A* in which just as in conventional A*, the search space is discretized and a graph is imposed on the grid with centers of cells acting as neighbors in the search graph. However, unlike traditional A*, the hybrid-state A* associates with each grid cell a continuous state of the vehicle.

This method is similar to the Field D-star method [6], which also tried to address the challenge of an A* agent being only able to visit the gird cell centers. However, the main contrast between the methods is that Field D* is limited to piecewise-linear paths, but hybrid A* is not. Since it uses the continuous kinematic car model when expanding the nodes, the loci produced by hybrid A* are assured to be 'drivable'.

There is a lot of literature on mathematically admissible heuristics for the A* algorithm. One very interesting method was to use an obstacle-sensitive cost function based on the Voronoj graph of the free space of the vehicle [14]. After matching the target position to the closest point on the Voronoi graph, Dijkstra's algorithm is used to calculate the shortest path distance to the target position for every point on the graph.

Using such a heuristic can help avoid the path planning getting stuck in dead-end configurations as the constructed Voronoi graph will faithfully describe the entire topology of the free space. For this reason, we initially decided to go with a heuristic that was based on inspiration derived from the above method.

The hybrid A* algorithm is guaranteed to produce drivable paths, but it is common to see it generating paths that require unnecessary steering actions. It is common in the literature to smoothen the generated set of waypoints through post-processing techniques to ensure a locus that gives a higher degree of comfort and safety. Particularly, we focused on a smoothing algorithm as described in another paper by Dolgov and Thrun [3].

For a given set of waypoints, they compute a metric based on four terms involved with different aspects of the path. There is a term that penalizes collisions with obstacles, a term that penalizes high curvatures, a term to ensure even spacing of waypoints, and a term to helps avoid obstacles. Since our configuration space and search already had plenty of safeguards against collisions with obstacles, we made use of only the terms in the metric that were not involved with obstacle avoidance for simplicity of implementation. This was followed by a gradient descent algorithm that shaped the path appropriately using a fixed number of iterations to ensure run-time consistency [2].

It is quite common in robotics literature to use a PID controller for path following once a set of waypoints have been generated. In the real world, tuning the PID controller to achieve optimal parameters is a major concern, especially for relatively unsophisticated path-finding algorithms. For instance, we came across a paper [9] in which a PID tuning is proposed, based on basic control tools, which takes into account the robustness of the closed-loop system. In our case though, we found that this was unnecessary since the path generated by the algorithms we implemented was traversable for the most part except for sharp curves and U-turns.

We came across a paper that used obstacle detection during planning time to modify the costs to a lower value in places in the free space that necessitated sharp curves and U-turns [12]. We borrowed certain ideas from this method to create a much more simplified system in which the car slowed down in preparation for upcoming sharp curves and U-turns during path execution.

The path planning for the drone was done in a similar way to the car. But we were using a dynamic point drive model to ensure that we took into account the agility of the drone [1]. One particular challenge that we faced with the drone was its low mass and the absence of drag. The low mass meant that almost all of the momentum for the dynamic point agent would show up in its velocity, making the agent oscillate around the desired path erratically. This is made worse by the absence of drag which also makes it hard to execute turns as there is nothing to sustain the centripetal force. In the real world, UAVs also face complicated terrains or highly dangerous regions with anti-air threats, disturbances like wind cause the UAV to deviate from the original path. A popular technique is to make use of a PID controller modified with a feedforward term to correct for such oscillatory disturbances.

We came across such a paper [11] that used cubic splines for smoothing the waypoint information followed by a PID controller with feed-forward for tight path following. The cubic splines ensure that the path is continuous and differentiable at all points, thus enabling the computation of curvature at very high resolutions. In most PID controller-based systems, the guidance command is given by only the PID controller. But the guidance command of this study includes the feed-forward controller which takes into account the acceleration caused by the curvature of the path. Because the curvature of the desired path is a known quantity, the feed-forward term cancels the acceleration due to the path curvature.

Another interesting approach was to use artificial forces programmed into the path execution to help the UAV follow a tight path [13]. The paper that used this method presented a virtual-force-based guidance law for path following and obstacle avoidance. They introduce a virtual spring force and a virtual drag force designed for straight-line following. They also add virtual centripetal force designed to counteract the influence of the curvature of the planned path. The most impressive feature that this paper employed in our opinion was the addition of an extra virtual repulsive force that pushes the vehicle away to move around obstacles. Inspired by this paper, we also made use of artificial physics to aid the drone in following a tighter path.

3 Proposed method and Implementation

In this section, we will discuss the proposed method and all of the stages that the car and the drone have to take to reach the goal on a given map. The general steps to autonomous driving and decision-making, differ slightly from other research like in [10], as in our problem the environment is static. Therefore, the main steps can be summarized into three main components: path planning and obstacle avoidance, post-processing (car), and vehicle control.

A Path Planning and Obstacles

Path planning varied for the car and the drone, as they have different kinematic models. The drone can move in any direction without having to change the orientation (i.e. it is omnidirectional), whereas the car is restricted to its orientation and the maximum rotation of the wheels.

For the drone, using the ordinary A* path planning algorithm was sufficient, especially given that the map we were using was already discretized into a grid. We implemented the algorithm to branch in 8 directions around itself at a constant distance (see 1 (left)). The heuristic that was used was the Euclidean distance to the goal, in the world frame, which is used to sway the algorithm towards the goal faster rather than exploring the whole search space blindly.

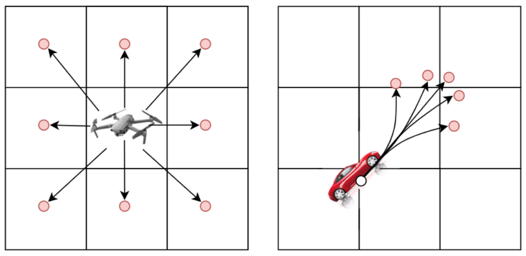

Figure 1: A* Path Planning for the drone (left); Hybrid A* Path Planning for the car (right)

On the other hand, for the car we used an extension of the ordinary A*, called Hybrid A* [4]. The main difference is that this algorithm considers the kinematic model of the car when branching out. Thus, instead of branching out in every direction, the branches will be generated at a given steering angle (α), where $\alpha \in \{-40°, -15°, 0°, 15°, 40°\}$ (see Figure 1 (right)). These angles were chosen to make either big or small left/right turns or to go forward. Just like with the drone, we defined a constant distance (d), and with that, we were able to calculate the new position and orientation of the car at the branch:

$$\beta = \frac{d}{L}\tan(\alpha), \qquad (1) \qquad\qquad R = \frac{d}{\beta}, \qquad (2)$$

where L is the length between the wheels, β the turning angle, and R the turning radius. With the turning angle from Equation 1 and the turning radius from Equation 2, the new positions can be found:

$$x' = \begin{cases} x - R\sin(\theta) + R\sin(\theta+\beta), & \text{if } |\beta| > 0.001 \\ x + d\cos(\theta), & \text{otherwise} \end{cases} \qquad (3)$$

$$y' = \begin{cases} y + R\cos(\theta) - R\cos(\theta+\beta), & \text{if } |\beta| > 0.001 \\ y - d\cos(\theta), & \text{otherwise} \end{cases} \qquad (4)$$

$$\theta' = (\theta + \beta) \bmod 2\pi. \qquad (5)$$

Notice that if the steering angle α is 0, the resulting turning angle β will be 0 too, meaning we only move forward without turning.

Finally, for both of the cases, obstacle detection has to be taken into account. Given that all the obstacles are static, this step can be done in the planning stage. The final approach that we decided to use was checking dynamically when new branches are generated, whether the vehicle's bounding box is overlapping any of the obstacles in the expected orientation at that point. It might not be the fastest approach, however, we found that it does not add a lot of time during planning, and it works well. Furthermore, we decided that inflating the size of the vehicles would aid us in generating paths that are more careful around the obstacles, especially when turning. The resulting paths that were generated are shown in Figure 2 (left).

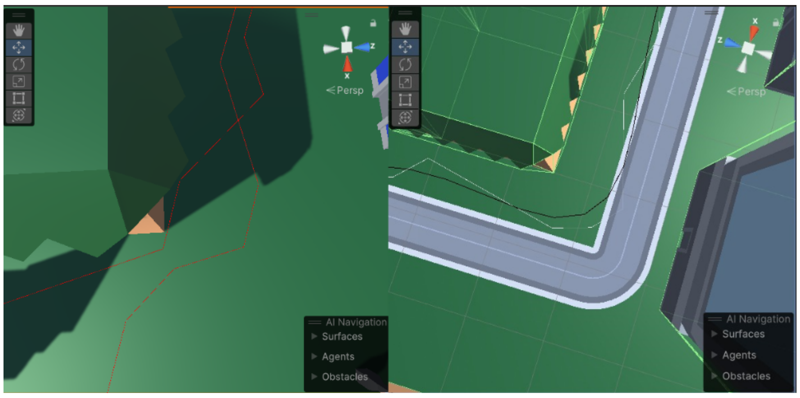

Figure 2: Difference when running Hybrid A* with and without obstacle detection + vehicle inflation (left); Difference between running Hybrid A* with and without smoothing, where the black curve is smoothed out and the white curve is not (right)

B Post-Processing

After path planning for the car is complete, we have a traversable and drivable path. However, the path produced is not fully optimal, as it still contains several unnatural steering motions due to only a discrete number of steering angles that are being considered. For that reason, we smooth out the curve to get a more optimal solution, similar to the approach proposed by Dolgov and Thrun [3]. The algorithm is a gradient-based approach, which uses a cost function composed of curvature (P_c) and smoothness terms (P_{smo}). The gradient descent method aims to minimize the following cost function:

$$P = P_c + P_s. \tag{6}$$

Firstly, P_{cur} is used to upper bound the instantaneous curvature of the curve for all of the vertices ($\boldsymbol{x}_i = (x_i, y_i), i \in [1, N]$). To calculate it, the following formula is used:

$$P_c = w_c \sum_{i=1}^{N-1} \sigma\left(\frac{\Delta\phi_i}{|\Delta\boldsymbol{x}_i|} - \kappa_{max}\right) \tag{7}$$

There are various components to this formula, so let's break them up one by one. w_c is the weight that is in charge of controlling how much impact this term has on changing the path. When iterating through all of the nodes, σ is used to penalize when the derivations from the path are too high, and κ_{max} is a constant that serves as the maximum allowed curvature. Moreover, $\Delta\phi_i$ checks the change in the angle between the tangent line and the curve at the vertex i. And lastly, $\Delta\boldsymbol{x}_i$ represents the displacement vector between the points i and $i-1$.

The second term, P_s, evaluates the smoothness of the vertices. It is computed by checking the displacement, Δx_i (as was described for the curvature term), between two vertices. The vertices which are unevenly spaced out, or have a change in direction will be assigned a higher cost. The complete formula is given as:

$$P_s = w_s \sum_{i=1}^{N-1} (\Delta\boldsymbol{x}_{i+1} - \boldsymbol{x}_i)^2, \tag{8}$$

where w_s is the weight that is in charge of controlling how much impact the smoothing cost has on changing the path. Figure 2 (right) shows the smoothed-out path compared with the normal Hybrid A* path.

C Vehicle Control

Just like with path planning, the vehicle control varies between the car and the drone pretty substantially. For starters, the car uses a PID controller, whereas the drone only a PD controller. The P is used to compute which way to steer based on how far away from the expected curve we are, D is used to smooth out the motions by seeing how quickly the directions are being changed, and I is used to correct any long-term errors by tracking the deviations. Our approach uses constant terms for all of the constants, and we calculate the steering angle (α) and acceleration (a) for the car as follows:

$$\alpha = w_\alpha \frac{90^\circ - \angle(\boldsymbol{V}_R, \boldsymbol{a}_d)}{\alpha_{max}}, \tag{9}$$

$$a = w_a \frac{\boldsymbol{a}_d \cdot \boldsymbol{V}_F}{|\boldsymbol{a}_d|}. \tag{10}$$

In Equations 9 and 10, $\angle$ denotes the function calculating the angle between two vectors, $\boldsymbol{V}_R$ and $\boldsymbol{V}_F$ denote the right and forward vectors of the car, respectively, and ws represent the sensitivity of acceleration and steering. Lastly, a_d stands for the desired acceleration and it is computed as $a_d = k_p * E_p + k_d * E_d + k_i * I$, where E stands for the positional and velocity errors (in that order), and I for the integral.

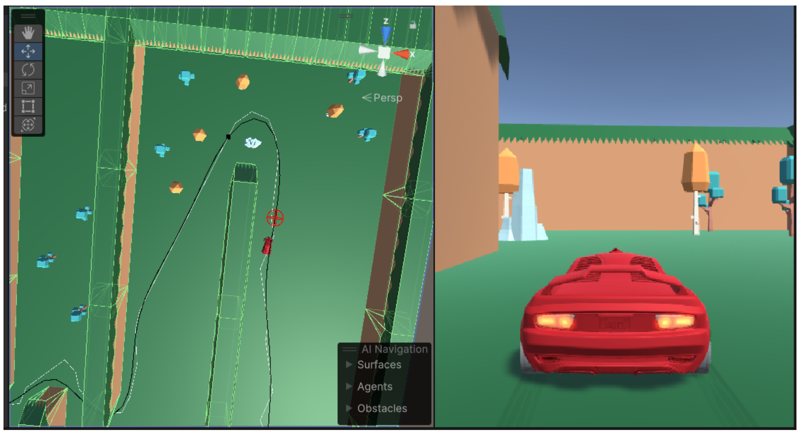

Figure 3: The black pointer shows the look-ahead point, and the red wire sphere shows the current next waypoint on the path that the car is aiming for. It can be seen on the right, that the car breaks, as it recognized that the there is a sharp turn.

Returning to w_a from Equation 10, updating this in real-time, based on the upcoming path is the main reason why we can follow the desired path very precisely and relatively quickly as well. To be able to accurately adjust this sensitivity, we employed an idea where the car will look a couple of nodes ahead so that we can see in time whether a turn is coming, and how sharp it is. Figure 3 shows what it looks like during runtime. This forward point is not set either, but it is dynamically decreased or increased based on some factors. In general, there are three main cases to consider: (1) no turn incoming, (2) mild turn incoming (100°), and (3) sharp turn incoming (> 120°). The angles are determined by comparing the predicted angle during the path planning stage and the current angle of the car (see Figure 3). For (1), we do not need to slow down or break, however, if the velocity increases, the look-ahead pointer will also increase; this way the car will have more time to react when a turn comes and it's going fast. For (2) we simply decrease the acceleration sensitivity slightly (from 1.0 to 0.8). We do not need to slow down more than that because: firstly, the car follows the path very accurately due to Equations 10 and 9, and secondly, the smoothed-out curve makes the arches bigger. However, for (3), if the velocity is too high, we will break to decrease it, and otherwise, we will go slowly (with $w_a = 0.5$) until the turn ends. During the turn, the look-ahead point will also be decreased temporarily, as coming out of the turn we will have a small velocity, meaning we are allowed to build it up a little bit first.

The drone uses somewhat a similar approach with a couple of simplifications. The ws for horizontal and vertical acceleration sensitivity are set to being constant. We noticed that due to the lack of a drag force in the environment it was difficult to execute turns and the drone's low mass made it oscillate around the defined path. To make the path following stable, we introduced an artificial drag into the rigid body (see Figure 3). With this, we were able to slow down acceleration, which in turn, made the steering of the drone much more controllable and precise.

4 Experimental results

4.1 Experimental setup

We used Unity (version 2022.3.15f1 LST) to run the simulations of the car and drone models. The script was written in C#. We used a drive distance set by the grid cell size of the domain multiplied by a distance splitting factor to ensure that the agent does not end up in the same cell after a node transition. We set a goal proximity threshold of 10 units of Euclidean distance for the convergence of the A* and Hybrid A* algorithms. The Box Collider objects of both the car and the drone were inflated by 1 unit while path planning to eliminate risks of collisions. For smoothing, we used a maximum acceptable curvature of 2 units and gave 40% more weight to the term in the smoothing metric that penalized high curvatures. For the PID controller in both the car and the drone models, we used the following parameters for all terrains: $k_p = 3$, $k_d = 1$,$k_i = 0.5$. We used a waypoint Euclidean distance threshold of 5 units for the car and 7.5 units for the drone. Once the agent falls within this circle around the waypoint, it shifts to the next waypoint in the path.

4.2 Analysis of Outcome

The implemented methods yielded a smooth and traversable path for the car model and it successfully reached the goal. In this section, we will focus on what in our opinion was the most challenging terrain, to point out the features of our model (see Figure 4).

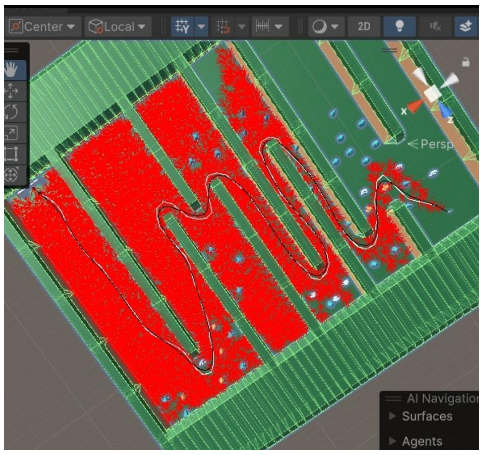

Figure 4: Visualizations on Terrain B – the planned path is in white, the black locus denotes the smoothened path, and the red is the exploration carried out by the Hybrid A*.

As it can be seen in Figure 4, the smoothened black curve is an improvement to the white path found by Hybrid A*. This is especially noticeable near the sharp curves where the black curve suggests steering with a larger turn radius (and thus lower curvature) to the car model. It is also worth pointing out that due to the smoothness term in the metric, the waypoints are evenly spaced and avoid unnecessary steerings for the most part. The car navigated all the terrain safely without any collisions and reached the goal in a reasonably good time.

When we compare our results with that of other groups in Table 1, we can see that other groups tended to perform better on terrains A, D, and C which did not involve sharp turns and instead had more straight-line segments. We performed better than most on terrains E and B which was riddled with hairpin curves. We were expecting this since we had been extra careful to make sure that the agents were performing well on sharp turns. In our opinion, there were two main reasons that group 4 was performing very well across all terrains. They had used Catmull-Rom splines to generate curves with a large ('smoother') turn radius so that they maintain speed on the turnings (similar to the effect that the curvature term had in our gradient descent program). Furthermore, they had the idea of combining it with linear interpolation to produce straight paths on milder turns, meaning that they were able to go fast on straighter curves. This combined with the very clever idea of having an adaptive parameter tuning for the PID controller based on the visibility map density meant that they could execute straight paths faster.

	Vehicle									
	Car					Drone				
Gr.	Terrain									
#	A	B	C	D	E	A	B	C	D	E
4	28.2	37.0	10.6	26.6	35.1	38.2	42.0	12.3	38.1	40.24
8	30.3	39.1	10.3	28.3	36.2	37.2	45.9	11.5	37.0	42.2
21	24.0	37.2	9.5	19.2	35.1	41.1	56.3	12.1	40.6	46.4
19	32.5	36.1	9.1	32.6	33.8	39.8	52.1	12.4	39.8	43.2
15	33.9	38.5	11.1	37.1	33.56	37.8	46.6	11.8	38.3	44.0
27	**33.2**	**39.3**	**11.4**	**33.7**	**33.64**	**40.0**	**46.3**	**13.14**	**41.1**	**42.3**

Table 1: Leaderboard for the top 6 groups based on the sum of the times across all the terrains, both for the car and the drone. Our solution was bolded.

Group 21 used a RRT* path planning, but what stood out was the fact that they used a cost function that was based on duration rather than the distance. This very smart trick allowed them to automatically find the path that required the least amount of time in theory. We feel that if we had used the velocities that we were already storing in our nodes in computing the duration cost for a node transition, then perhaps we too could have achieved such good results. For the case of the drone, it is worth noting that Group 21 cleverly removed the component of velocity that was causing the drone to veer off course, instead of applying a complete damping effect like we did. This ensured that their drone could move faster in straight paths (that is if it were going in the correct direction). Group 19 had solutions that produced results that were mostly similar to ours. But the one thing that they did better in our opinion was to perform smoothing based on line of sight and interpolation. This algorithm (also used by group 29) allowed to get a traversable path made of mostly straight line segments with a minimum amount of breakpoints. This is the reason why they perform marginally better than us in most of the terrains. Group 15 had very similar results to us. But we were impressed by the sophistication of their PID controller which in they implemented a spheric lookahead, which calculates the angle of intersection between the planned path and the vehicle's forward motion. By projecting the path onto a sphere centered on the vehicle, this method provides a comprehensive view of the upcoming terrain, enabling more precise navigation. This combined with the idea of damping the velocity orthogonal to the planned motion similar to group 21, meant that they performed better than us in certain terrains.

5 Summary and Conclusions

The overall results from this project were satisfactory, for both the car and the drone. To summarize the method from Section 3, firstly the path planning algorithm is being run on the given terrain to find the most optimal path. For that, we are using A* and Hybrid A* algorithms for the drone and the car, respectively, and on top of it, when generating new branches in these algorithms we check if there will be a collision with the obstacles. Afterwards, for the car, the path is smoothed out. For the drone, it was not necessary, and the results showed that it followed the path very well. Finally, vehicle control is implemented, where the car follows the waypoints on the path and looks ahead to check if it needs to slow down for a turn. On the other hand, we apply a constant artificial drag to the drone to make the maneuvering smoother and more controllable. Therefore, no matter the environment that is given, a traversable path to the goal will be discovered, if one exists. We believe that the main contribution of this project is creating a vehicle controller that can very precisely follow a path, regardless of what it is, at a relatively high speed. Running the algorithm on previously unseen test Terrains, showed that it can adapt and still perform very well. Nevertheless, that is not to say that there are no possible improvements to be made. As we have analyzed in Section 4, there are a lot of interesting concepts that other Groups have implemented, which would have helped us improve our results further. Given more time, the main improvement that this research would need to improve the results from Section 4, would be to speed everything up (see Appendix). Some of the main ideas would be: speeding the car up when there are no turns, being less conservative when turning (as that is where we lose the most speed), and having a more dynamic movement model when flying the drone (just like we had for the car), as applying constant drag is not ideal for speed. We also tried implementing extra heuristic functions, specifically a vector field, but the results did not yield much of an improvement in terms of path planning. Finally, this research also shows a big promise in autonomous driving (for now with a static world, but another possible extension is to use something like Delayed D* [5] to adapt to a dynamic world), as having a car that can follow the path very smoothly and carefully, without swaying from the given path, and doing it at a relatively high speed, is a big achievement. The reason being is that if the car moves away from the path even slightly, it could cause a crash. To conclude, despite there being a possibility to improve the results even further, given the time frame and the fact that we had to work on both the drone and the car, which did differ to an extent in certain aspects, this project was still a big success.

References

[1] Xiong Bai, Haikun Jiang, Junjie Cui, Kuan Lu, Pengyun Chen, and Ming Zhang. Uav path planning based on improved a* and dwa algorithms. *International Journal of Aerospace Engineering*, 2021:Article ID 4511252, 12 pages, 2021.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[3] Dmitri A. Dolgov. Practical search techniques in path planning for autonomous driving. 2008.

[4] Dmitri A. Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29:485 – 501, 2010.

[5] Dave Ferguson and Anthony Stentz. The delayed d* algorithm for efficient path replanning. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2045–2050. IEEE, 2005.

[6] Dave Ferguson and Anthony Stentz. Field d*: An interpolation-based path planner and replanner. In Sebastian Thrun, Rodney Brooks, and Hugh Durrant-Whyte, editors, *Robotics Research*, pages 239–253, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4:100–107, 1968.

[8] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[9] Julio E. Normey-Rico, Ismael Alcalá, Juan Gómez-Ortega, and Eduardo F. Camacho. Mobile robot path tracking using a robust pid controller. *Control Engineering Practice*, 9(11):1209–1214, 2001. PID Control.

[10] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[11] Ihnseok Rhee, Sanghyuk Park, and Chang-Kyung Ryoo. A tight path following algorithm of an uas based on pid control. *Proceedings of SICE Annual Conference 2010*, pages 1270–1273, 2010.

[12] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William “Red” Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[13] Xun Wang, Libing Cai, Longxing Kong, Binfeng Wang, Shaohua Huang, and Chengdi Lin. Path following and obstacle avoidance for unmanned aerial vehicles using a virtual-force-based guidance law. *Applied Sciences*, 11(10), 2021.

[14] J. Ziegler, M. Werling, and J. Schroder. Navigating car-like robots in unstructured environments using an obstacle sensitive cost function. In *2008 IEEE Intelligent Vehicles Symposium*, pages 787–791, 2008.

Appendix

We would like to use this part to point out some flaws with the normal Euclidean heuristic. It can be seen in the red portions of Figure 4 that a lot of exploration is being done in the areas far from the turning junctions and near the walls. This is because the Euclidean heuristic underestimates the cost to the goal at the nodes near the walls (as it is agnostic to obstacles), which in turn encourages the A* algorithm to explore those regions at a higher resolution. This results in the planning algorithm taking a longer time to converge.

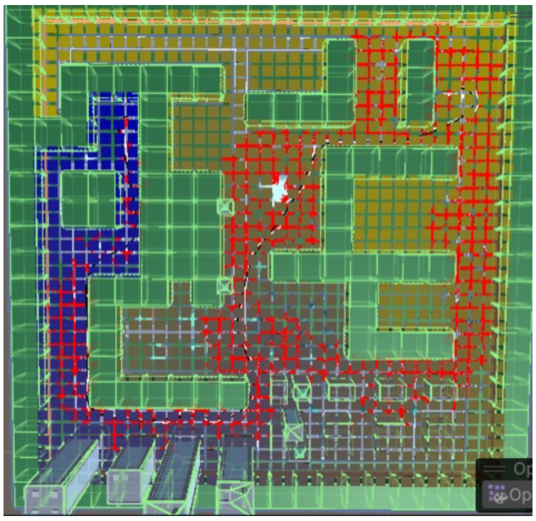

Figure 5: The yellow colored blocks have a lower cost than the bluer blocks. The red is the exploration carried out by the hybrid A*

To combat this issue, we tried to implement a flow field-based heuristic which performed a normal A* at every node to estimate the cost-to-go at each node. We tried a combined heuristic by taking the maximum of the flow field and the Euclidean heuristic and we saw that this discouraged explorations near the walls leading to a smaller convergence time. You can notice that in Figure 5 the red searching portion avoids the area near the walls that are dead ends all together. However, we ran into technical issues while implementing this on test terrain B, and as a result we did not use this in the final version of our project.