Towards cooperative collision avoidance for the Voyager unmanned ground vehicle: practical implementation and demonstration

Thando Mtimkulu[1], and *Jacobus A. A.* Engelbrecht[1]

[1]Department of Electrical and Electronic Engineering, Stellenbosch University, South Africa

Abstract. This paper presents the practical implementation and testing of a multi-agent cooperative collision avoidance system on the CSIR's Voyager mobile robotic platform. The system consists of a long-term global path planner and a short-term local path planner. Multiple robotic platforms execute their individual global paths while broadcasting their intent and cooperatively predicting short-term collisions. If a collision is predicted, the platforms cooperatively and sequentially plan and execute short-term collision avoidance paths, returning to their global paths. The effectiveness of the system is verified both in simulation and with practical tests using the physical Voyager robotic platform.

1 Introduction

In recent decades, research in autonomous vehicles has progressed extensively in both academia and industry. Within the latter, it serves many interests and benefits centred around safety or efficiency. It can be a complex control problem with a variety of possible objectives, contexts, hazards, knowns, and unknowns.

As more of these vehicles are adopted, more of them will pursue unrelated goals in commons spaces that may clash in execution. Autonomous vehicles themselves become hazards to one another. However, mutual consideration by those agents amongst themselves could mitigate that. The work presented in this paper aims to examine one approach to this.

1.1 Goal

The goal of this research is to demonstrate multi-agent cooperative collision avoidance on the Voyager unmanned ground vehicle (UGV), which was developed, provided, and funded by the CSIR. In doing so, we assess the approach of the chosen solution in the less-than-ideal real world and note factors significant in realising the system from simulation. This solution would allow several UGVs in a previously mapped environment to autonomously plan routes to given independent destinations. They would leverage inter-vehicle communication to cooperatively resolve collisions predicted in the execution of their individual paths.

The cooperative collision avoidance system is deemed successful if all vehicles reach their destinations without collisions. In addition, the performance of the system is evaluated

based on criteria and metrics like deviation from the planned path, destination, or schedule, as well as time taken and number of attempts to perform the path planning.

For this paper, "the platform" will refer to the vehicle, its hardware, and their associated software drivers. The "system" will refer to the cooperative collision avoidance software solution introduced onboard.

1.2 Literature

The field of autonomous navigation is very well developed in the breadth of literature available. Many works cover several of the issues and questions involved at various levels and by varying approaches. The pertinent issue of cooperative collision avoidance features many smaller or specific problems within that need to be solved before or alongside the main problem.

Betz et al. [1] cover many of these problems in their survey targeting autonomous racing, as well as the approaches and progress made to address them. They consider an autonomous vehicle as a pipeline of perception, planning, and control as shown in Fig. 1. The components of the pipeline are used to categorise the research discussed as a helpful breakdown of typical autonomous vehicles.

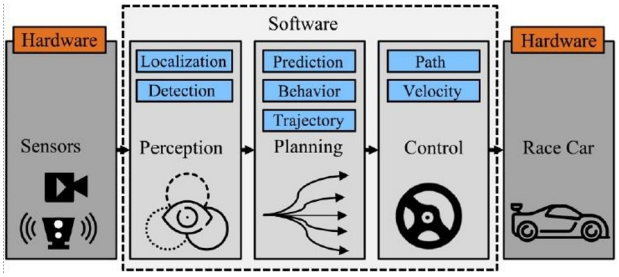

Fig. 1. The autonomous driving pipeline [1].

Flowing through the pipeline, robot perception is one of the first sub-problems encountered. One aspect of it, localisation, is focused on deducing the pose of the vehicle from sensor measurements of its state or that of the environment. In this regard, much of the localisation research surveyed featured Extended Kalman Filters (EKFs) or Graph SLAM (Simultaneous Localisation and Mapping) implementations. The former uses models and measurements to find the optimal estimate of the vehicle's pose, given the uncertainty of each. The latter formulates localisation as a graph problem, offering an estimate of the current pose as part of its solution and a map of the traversed environment. From the research they tabulated, one can see that even though implementations are functionally interchangeable to the system, their performance and sensor requirements impose constraints on the systems capabilities. These should be considered when choosing between them.

Control of an autonomous platform is another sub-problem to address. Specifically, how the vehicle should move through planned positions at planned times. There are many types and combinations of controllers used to do this. Of these, there are two ubiquitous types: geometric and optimal controllers. Geometric controllers, such as Pure Pursuit or the Stanley controller, use the geometric relationship between the vehicle and a reference path to produce a control input such that it eventually converges to the path. Optimal controllers, such as Model Predictive Control (MPC), use knowledge or models of vehicle states to solve for control inputs that bring the vehicle to the path such that some optimal cost or objective is achieved. Which of the two is appropriate depends on what is required of the control section and the resources available to it. Furthermore, each can be improved or specialised per

application. For example, AbdElmoniem et al. [2] add model predicted states to the Stanley controller.

Planning, the focus of this research, is in the middle of the pipeline. Here, motion planning boils down to supplying the control section with what it needs to get from its initial state to some specified target state based on input from the perception section. For collision avoidance, this means planning a route that does not collide with obstacles and reaches the destination. Radmanesh et al. [3] give a comparative overview of the approaches and algorithms used. These vary widely from constructing and solving potential fields to graph search techniques, many of which can be applied in combination to meet requirements. For example, Barraquand and Latombe [4] use a potential field for motion planning through obstacles, Roberge et al. [5] compared the performance of the Genetic Algorithm and Particle Swarm Optimisation for UAV path planning, and Desaraju et al. [6] achieve decentralised cooperative planning and collision avoidance with Rapidly-exploring Random Tree (RRT).

Hughes [7] also developed a system capable of cooperative collision avoidance in unmanned aerial vehicles (UAVs) with a low rate of failure in simulation using graph search techniques. Their approach split their motion planning into route planning and collision avoidance, with the latter further split into collision prediction and resolution. For the route planning, A* search was used; specifically, the graph searched consisted of the edges of a Voronoi diagram of the map as shown in Fig. 2. This provided the shortest path of maximum clearance from known static obstacles, giving space for evasive manoeuvres performed while travelling.

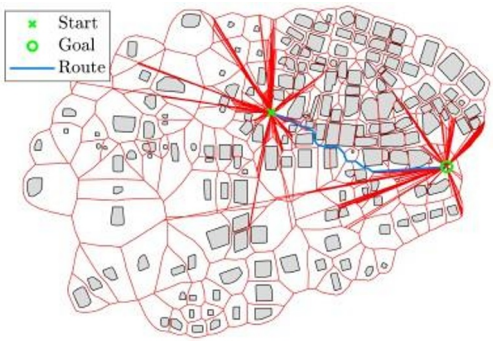

Fig. 2. Voronoi diagram route planning [7].

For conflict resolution, Hughes used Dijkstra's algorithm on a sprawling tree of horizontal and vertical evasive manoeuvres costed with respect to their deviation from the planned path. Their cooperative strategy, like that of Desaraju et al. [6], was decentralised, but in contrast was in sequential fixed priority with no way to alter the paths of other vehicles. This means an agent could minimise its own cost but not the cost of the collective, leading to less than optimal but real-time solutions. Radmanesh et al. [3] also note that graph search algorithms like Dijkstra and A*, though they provide optimal or near optimal solutions if they exist, grow in computation time with the size of the problem and graph. However, because agents only plan for themselves, this growth is expected to be linear as the number of agents scales.

2 Methodology

In the development of the system, several sub-problems are encountered. The methodologies discussed are introduced to address them. In collision avoidance there are sub-problems of localisation, control, obstacle detection, collision detection, and modelling, all under the platform's constraints.

2.1 System overview

Overall, the system's functional layout follows that of the autonomous driving pipeline (Fig. 1). Selected sensors feed into perception-related subsystems that pass information to those related to planning. The interface is meant to fulfil the functional prerequisites of the planning section like a map or current pose. Planning directs control in an analogous way, providing and updating references to be tracked. There are three important notes regarding the pipeline layout.

Firstly, supporting each of the pipeline sections are the platform drivers that manage system start up, sensor management, timing, fault conditions, and communications. Secondly, said communications will be considered a component of the planning section despite being concerned with sensing other vehicles as it forms an integral part of the system's cooperative planning. Lastly, the pipeline's flow is not necessarily the same as the system's operational flow. The following Fig. 3 can be contrasted with Fig. 1 to see why.

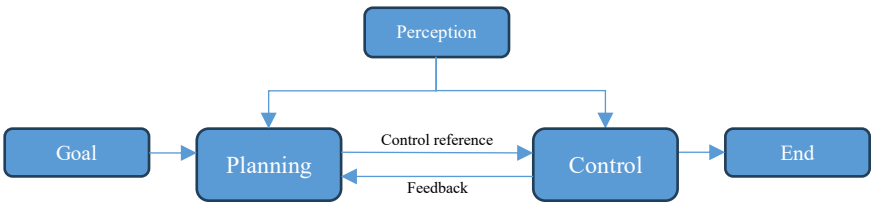

Fig. 3. Operational flow of the system.

Missions undertaken by the system start with the reception of a goal by the planning section. Once a route is planned, the control section is prompted to action and updated along the way as needed. In other words, where data flows from Sensors to Race Car in Fig. 1, the system's state alternates between planning and control until the goal is reached.

2.2 Platform

The Voyager UGV is a differential drive vehicle of moderate size (approximately 0.8x0.7m) developed for research in the control of autonomous vehicles. In this work, it is a model system meant to represent any arbitrary ground vehicle. It is equipped with many of the sensors UGVs could have access to. The IMU, 360°x90° lidar, and wheel encoders are the most significant for the work to follow, though an RGB webcam and GPS are also onboard.

The vehicle has two drive wheels that can rotate independently for differential drive control and two castor wheels for stability. Differential drive vehicles move by commanding different speeds for each wheel to change their linear and angular speeds; however, the platform's software drivers abstract the motors away so only a linear/angular rate of the body needs to be commanded.

A computer is also featured onboard from which the system will run with access to the mentioned utilities and drivers. Communication and control of the platform is done in ROS2 Humble so consideration for compatibility with those structures is taken where needed.

2.3 Perception

In the development of the full system, the perception block addresses key sub-problems in avoiding collisions. More specifically, it needs to interface with sensors via ROS2 communication to localise within an environment. It further needs to generate or at least represent the environment (i.e. a map) for planning purposes.

The platform features its own pose estimation based on wheel encoder data; however, its accuracy wanes with distance travelled. To address this, IMU data, known to drift with time, is included to improve the odometry using a Kalman filter. The robot localization ROS2 package contains a platform compatible implementation of an Extended Kalman Filter (EKF) [8] that was used onboard.

Localisation of the filtered odometry and map generation, especially in test locations not previously mapped, is a problem SLAM (Simultaneous Localisation and Mapping) algorithms are oriented toward solving. It is a researched field with many solution approaches and implementations based on available sensors. For onboard environmental sensing, the platform has a webcam and lidar. SLAM implementations featuring lidar were given priority in choice due to the immediate availability of depth information from the lidar and its wider field of view. RtabMap [9] was chosen for its utilisation of 3D lidar, available implementation in ROS2, ability to work over multiple sessions, and map representations available. The flow of data through the perception section is shown by Fig. 4.

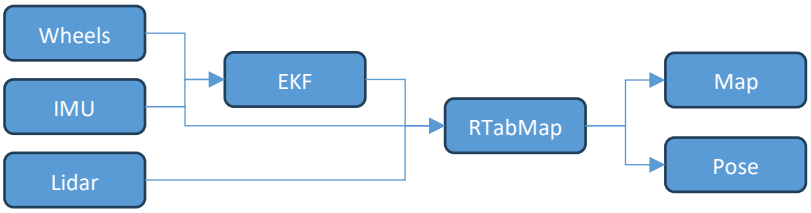

Fig. 4. Perception block diagram.

Here one can see how the IMU and encoders feed into the EKF that, along with the lidar data, feed RTabMap. On the right, what remains is a map and pose estimate therein.

ROS2 provides data structures for each that can be serialised and made available over topics subscribed to within a network. Key components of the pose estimate data structure consist of a time stamp, 3D position, and quaternion orientation. RTabMap provides a map in two data structures mainly, a 2D occupancy grid and a point cloud. The latter was chosen to preserve depth information for dependent processes and methods. Furthermore, the occupancy information of interest is expected to be sparse as densely occupied spaces may not be traversable at all. This means most of an occupancy grid's memory would be used to store nothing, making it less memory efficient with respect to occupancy than point clouds, despite point clouds typically having more points and an extra dimension to store. Memory efficiency could be improved by using a quad-tree or, when using point clouds, an octree.

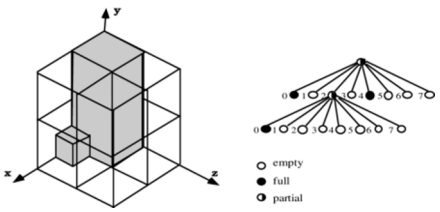

Fig. 5. Diagram of octree data structure [10].

Fig. 5 shows how occupied regions are stored as indexed eighth subdivisions of a tree structure. Given the geometric relationship between the tree structure and the space of the map, storing the point itself is redundant, reducing the memory used. The system used Point Cloud Library (PCL) methods and data structures for constructing and querying the map for information on static obstacles.

2.4 Control

On the opposite end of the system, the control block is largely concerned with the following of a structured collective of control references supplied by the planners, i.e. a plan. The platform's drivers abstract away control of the motors to control of the body; thus, the system only needs to be concerned with high-level control, namely path tracking.

A path can be defined in many ways depending on what the vehicle intends to navigate. It could be point-to-point or curvi-linear for example. The system is expected to generate and follow curvi-linear paths through general spaces. To do this, we represent paths as cubic splines, as they can produce straight lines or approximate arbitrary curves.

Cubic splines are piece-wise defined parametric curves that use cubic polynomials along each component to trace a path through space as a parameter variable varies between set knot values, usually 0 to 1. Using matrices, there are distinct types of cubic splines with characteristic matrices that change the format of a corresponding control point matrix it should be multiplied with. Hermite splines define a path based on its start position *P0*, end position *P1*, and tangent vectors *V0* and *V1* at those respective positions, all of which are row-vectors. Those are the main geometric qualities of interest for our purposes and were why this type is chosen.

$$\mathrm{P}(t) = \mathrm{T}(t)\mathrm{C}M \tag{1}$$

$$\mathrm{P}(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ V_0 \\ P_1 \\ V_1 \end{bmatrix} \tag{2}$$

Equation 1 shows an equation for a spline with parameter variable t in matrix form. C is the characteristic matrix that, when multiplied by the control point matrix M, produces a matrix of columns of cubic polynomial coefficients for each spatial component. Further multiplying with the left row matrix $\mathrm{T}(t)$ evaluates the polynomials at t, giving the point along the path associated with that value.

For a defined path, there are many path-following algorithms that vary by computational load, tracking error, and other behaviour. Between the two discussed in section 1.2, optimal controllers tend to offer closer path tracking but are more computationally intensive, running at lower update frequencies. Because the system requirements are only geometric, a geometric controller, the Stanley controller, is used. This saves processing power but makes the system vulnerable to errors or slips in localisation.

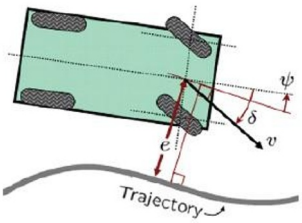

Fig. 6. Stanley controller frame and errors [9].

The Stanley controller is a geometric controller that gives a rectifying steering angle based on the heading and cross track error shown in figure 6 as ψ and *e* respectively. These errors are taken from the position and tangent direction of the closest point along the spline. Finding this nearest point involves subtracting the vehicles current position from the polynomials of each component before using the distance formula and solving for the parameter value at the minimum using NumPy or the GNU Scientific Library (GSL). Once the errors are calculated,

the control law of equation 3 is used to find the necessary steering angle δ for the vehicle travelling at speed v. K_{ct} and K_v are gains used to tune the controller's sensitivity to cross track error and low speeds.

$$\delta = \psi + \tan^{-1}\left(\frac{K_{ct}e}{v+K_v}\right) \quad (3)$$

The controller was initially conceived for control of a steering controlled vehicle. To apply it to the platform, the steering angle must be converted to an angular rate. A bicycle model was used as a simple analogue between the steering vehicle required by the control law and the dynamics of the real platform. The angular rate of a bicycle with a front to back tire wheelbase matching the platform, L_{wb}, is given in equation 4 and used to command the angular rate of the vehicle.

$$\omega = \frac{v\tan\delta}{L_{wb}} \quad (4)$$

With the spline path reference and Stanley controller, the vehicle will be confined to the path en route to the goal. This is sufficient for general autonomous navigation, but insufficient for collision avoidance of dynamic obstacles, as progress and the timing thereof become important. The vehicle should be in its expected position or progress at an expected time, so an in-track proportional controller was added to adjust the commanded speed of the platform such that it met a set departure and arrive time, assuming linear progress over that interval. Fig 7 shows the block diagram of this controller.

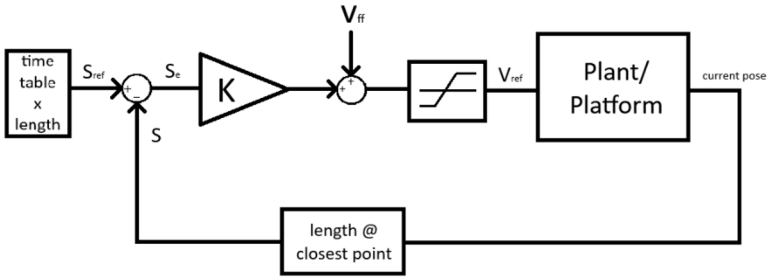

Fig. 7. In-track controller block diagram.

The proportion of the time elapsed within the spline's interval is combined with its length to obtain the expected in-track progress S_{ref}. Calculation of the length was done using GSL and SciPy numerical integration methods as the integral of tangent norms (root of sum of squares of component polynomial derivatives) need elliptical functions better approximated than used exactly. S_{ref} is compared to the length S travelled at the closest (current) point and multiplied with the tuned gain K to get the amount the prescribed mission speed v should be increased or reduced. The total recommended speed is then clamped to chosen limits to be forwarded to the Stanley controller and platform.

2.5 Planning

As the main functional component of this research, the planning section has many moving parts. To reiterate, the target system behaviour is autonomous cooperative collision avoidance. To accomplish this, the planning approaches of Hughes [7] were followed. Fig. 8 shows the internal configuration that entails.

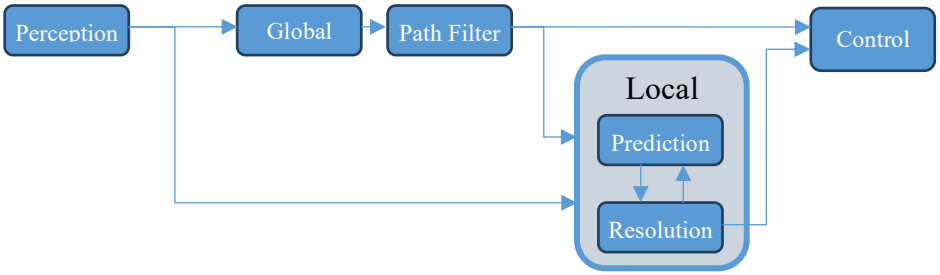

Fig. 8. Planning section block diagram.

On the left, one can see the interface of the perception block feeding into two planners marked Global and Local. The former implements the Voronoi A* route planner of Hughes whereas the Resolution component of the latter contains their evasive manoeuvre tree search and cooperative strategy. Within and around these blocks are other details that make system specific considerations or otherwise help realise the system.

One example of this is the Path Filter block that follows the Global. In the prior work, the Voronoi diagram and consequent roadmap was produced over the convex hulls of obstacles in a flattened slice of an environment tri-mesh, a different data structure than the point cloud octree used here. Where the Voronoi diagram of Hughes produced a curve between the side of one polygon and the vertex of another, the map's diagram will produce line segments between cloud points of the obstacles that approximate that curve. Geometric controllers may struggle to follow the jagged paths produced because of this, so the path must be smoothed. Conveniently, the polynomials that define the splines used by the control section are smooth functions, so simply deriving a suitable control reference to represent path points solves this problem. Least squares estimation is used component-wise to fit splines (i.e. polynomials) to the global plan points G_n (given n points). Equation 5 shows how the problem is set up based on Equation 1.

$$\begin{bmatrix} G_1 \\ \vdots \\ G_n \end{bmatrix} \approx \begin{bmatrix} T(0) \\ \vdots \\ T\left(\frac{i}{n-1}\right) \end{bmatrix} CM_{est} \quad \text{for} \quad i\epsilon[0,n) \tag{5}$$

On the left-hand side, we have the known plan points. On the right-hand side, CM_{est} represents the unknown least square error polynomial coefficients we want to solve for, with M_{est} being the control point matrix thereof. One further needs to choose the hypothetical parameter input values at which the left-hand side points occurred, constructing a larger matrix from stacking the $T(t)$ row-matrices. For simplicity, the points were distributed evenly in the parameter interval 0 to 1. M_{est} is found by multiplying the result of the Normal Equation, that solves Equation 5, by the inverse of C.

One spline might not be enough to closely represent the planned route so the least square estimation was wrapped in a binary search over the path points to find the most points that could be estimated by a single spline to within some specified tolerance. More or fewer points are attempted depending on the success or failure of the fit, respectively. This divides the planned route into splines that can be scheduled and augmented into full control references for that section.

Moving on to the Resolution block that, despite following the approach of Hughes, has two contrasting details. The first is the manoeuvre tree's cost function that would have been the cumulative deviation from the planned path. Given the concern and control of timing, the cost was adapted to the cumulative deviation from the planned motion, including progress. Fig 9 illustrates the subtle difference.

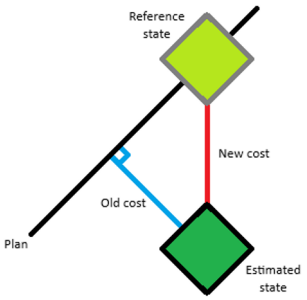

Fig. 9. Manoeuvre tree cost definition comparison.

Before one would use the shortest distance to the path. Instead, the distance to the vehicles expected position at that time is used. The expected position could be inferred from the control reference, but we use a model prediction instead to account for the vehicle's motion if or while away from the path.

The second contrasting detail is that a different set of evasive manoeuvres are defined, given the application is to a ground vehicle, not a UAV as in Hughes' case. Fig. 10 shows the new manoeuvres that are considered when resolving predicted collisions.

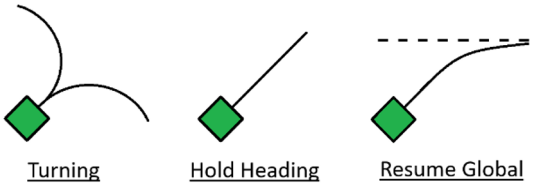

Fig. 10. Evasive manoeuvres.

Each manoeuvre is associated with a control point matrix definition that specifies the path followed by the controller during each manoeuvre. The Turning manoeuvres use circular arcs, the Hold Heading manoeuvre uses a straight line, and the Resume Global manoeuvre can use either a segment of the global plan or a least square estimate of the expected trajectory for that interval. This is packaged with the remaining control information and sent to the control section, beginning avoidance.

This local plan is also received by the Prediction block to update its model accordingly. As implied, this block generates a time stamped list of the platforms future positions based on the current pose, plan, and internal kinematic model of the controller. Collisions are predicted by checking these future positions for conflict.

The Prediction block also functions as the centre of the system's agent-to-agent communication as the generated list is also published to all other agents on a shared ROS2 topic to perform their own collision detection, making the beginning of the system's cooperative strategy. When a conflict is detected, the prediction block begins a protocol of communication to resolve the conflict. It implements random priority assignment in a decentralised fashion according to which participants will plan sequentially. Fig. 11 shows the structured communication involved between agents from the perspective of one.

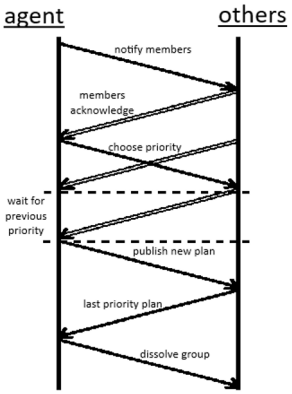

Fig. 11. Cooperative strategy protocol diagram.

The communication interface of an agent is shown on the left and the collective of other agents is shown on the right. Double line arrows indicate that the indicated exchange could be multiple messages. Starting at the top, relevant vehicles are notified of an impending collision, establishing a group once all acknowledge. Agents then randomly choose a priority, rerolling clashes. Only when the previous priority publishes its updated future trajectory does the current agent trigger its Resolution block. Once the agent publishes its updated future states, it waits for the update of the last priority vehicle before leaving the resolution group.

3 Results

As the system was developing, subsystems and dependencies were tested in simulation and then onboard. For practical tests, a nearby corridor was used for its size and constraints with unique surrounding features aiding in localisation. The results presented are all from experiments performed in this corridor and plotted from recorded data. Figs 12 and 13 show a picture and RTabMap occupancy map taken of the area to convey its layout. It has two double doors on either side of a long mid-section that were used as static obstacle constraints during testing.

Fig. 12. Test corridor

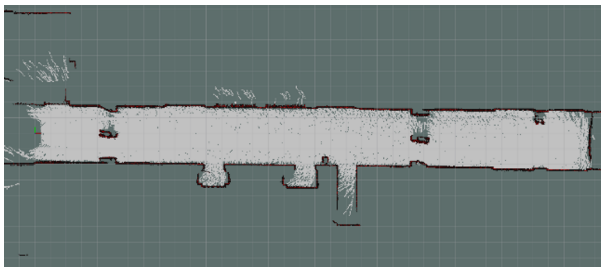

Fig. 13. Test corridor occupancy map.

Testing of the control section mostly focused on tuning the controller to the platform by trial and error. Control parameters included not only the Stanley and in-track controller gains, but a virtual steering limit that was also set to bound the vehicle's angular rate. A virtual angle of 60° was found to strike a satisfactory balance between keeping the system safe and taking advantage of the differential drive's agility. In Fig. 14 the vehicle was started at a right angle relative to the reference path in green to show the controller's response to a large initial

heading error. Fig. 15 repeats the test with the same reference path, this time starting with the vehicle offset by just over 1m to show the controller's response to a large initial cross track error. Having set K_{ct} to 0.5 and K_v to a small number (0.01) compared to the reference speed (0.3m/s), both figures show the vehicle return to, but slightly overshoot, the track. This response was deemed preferable to ensure that the controller was more aggressive in reaching the path and staying near it. Fig. 16 shows the displacement-time plot of the in-track controller test. For that test, the vehicle was given a long straight path plan but held back at the start, forced to catch up on the lost progress. This was successful as we see the blue vehicle trace join up with the green reference progress. A gain value of 0.5 was used as it kept the vehicle striving to keep pace at its maximum allowed speed of 0.5m/s until it was very close to the reference without overshooting it.

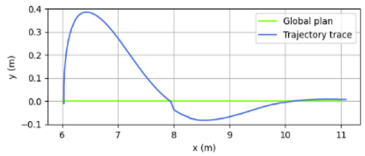

Fig. 14. Right-angle test plot.

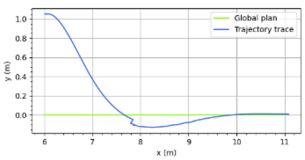

Fig. 15. Offset test plot.

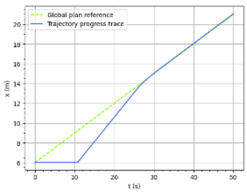

Fig. 16. In-track test plot.

The two planners were demonstrated in two separate tests. For the global planning test, the vehicle was positioned on one end of the corridor and given a goal that was on the other, with the problem being the two middle walls and doors in the way. The system would need to use the map point cloud and other resources to autonomously plan and move to the goal. Fig 17 shows a plot of the test results, including start, goal, Voronoi roadmap vertices, reference path, and a trace of the trajectory driven in the corridor. Therein, one can see the reference path board the road map through the doors, follow the vertices down the middle and off board straight to the goal. Key features like the straight line on/off boarding from the roadmap, following the Voronoi vertices through curves, and the path being overall conflict-free shows that the global planner operates as designed. The trace shows that the plan was successfully completed without incident, aside from the large S curve due to the reference switching logic following small paths in that area.

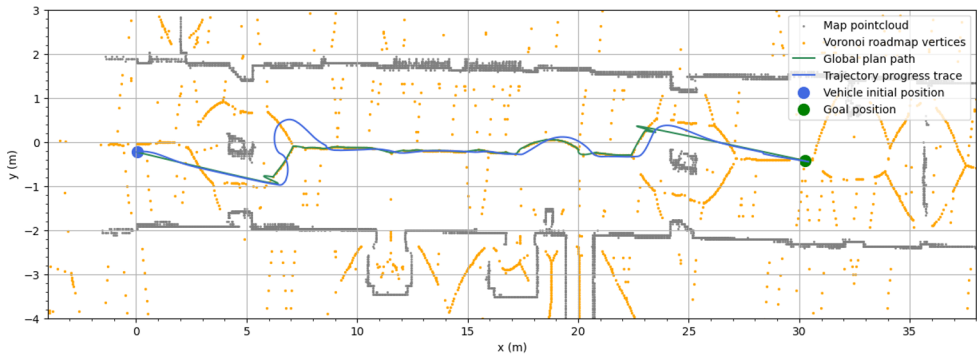

Fig. 17. Global planning test plot.

For the local planner testing, more specifically the cooperative avoidance, two virtual agents were added. In the mid-section of the corridor, they were all explicitly given references that set them on a course to collide one by one. The real agent was given the lowest priority, placing the burden on it to avoid both. The virtual agent's plans were biased/offset from that of the real agent for it to avoid to its right and left. To alleviate constraints in space, the virtual vehicles were permitted to pass through static obstacles as "ghosts". Fig 18's (a) to (f) give a

play-by-play through stages of the test. They show the exclusion zones (0.4m) of the vehicles and the trajectories they were projected to follow in the next 10s, as read from the data shared on the cooperative protocol topic. In (a) one can see the prediction of the real and nearest ghost agent extending toward each other, forecasting a conflict and triggering the resolution routine. (b) shows the longer conflict resolving local plans (20s with approximately 5s per manoeuvre) exchanged as part of cooperative planning. In (c), the plan was executed and did indeed resolve the collision. (d) to (f) repeat this in a turn to the left to avoid the second ghost car. (b) and (e) show planned evasive deviation that subsequently rejoins the global plan, reflecting the costing strategy of the manoeuvre search. Comparing (b) to (c) and (e) to (f), the vehicle follows its predicted trajectory very well making them reliable. Combined with the two successful avoidances, these show that the system was implemented successfully, and it behaves as designed.

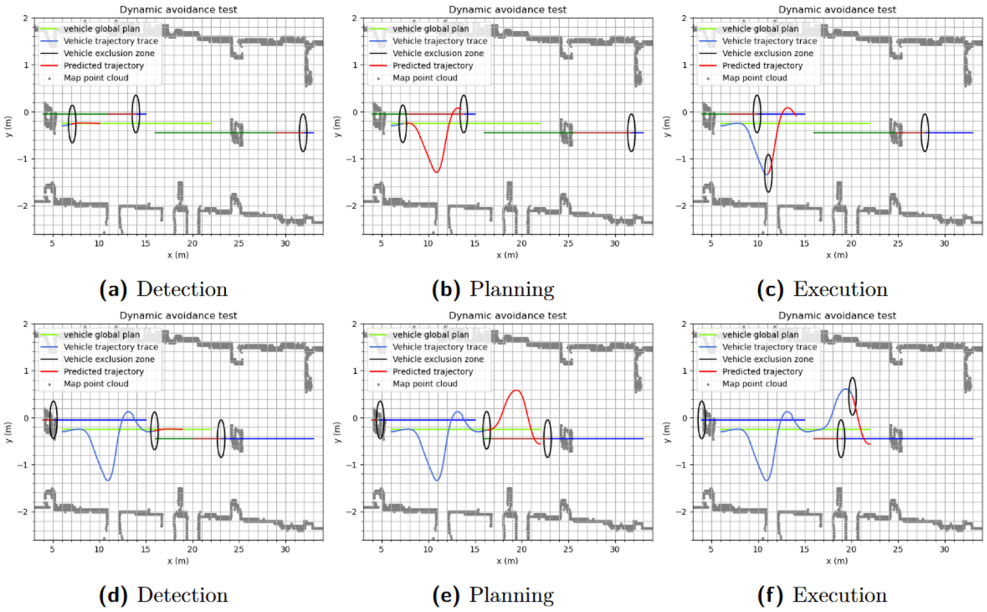

Fig. 19. Local plan test plot.

Fig. 18. The voyager UGV.

4 Conclusion

This paper presented research work done towards implementing cooperative collision avoidance on the Voyager UGV (fig 19). It uses the path planning and avoidance

methodologies of Hughes [7], planning long term routes using A* on the map's Voronoi diagram, and avoiding short term predicted conflicts cooperatively by taking turns finding optimal evasive manoeuvre sequences using Dijkstra's algorithm. These plans were followed by a modified Stanley controller to reach the goal.

Simulation was used as part of the development process before practical tests with the physical vehicle were done in a mapped corridor nearby. The global planner was able to allow the system to autonomously navigate into and out of partially obstructed sections using the Voronoi roadmap. Furthermore, cooperative collision avoidance for this autonomous system was shown working on the real vehicle avoiding two virtual agents. The success in local avoidance planning was dependent on both the order in which agents were prioritised and how well the manoeuvre options meshed with the complexity of the environment.

Latency in planning was not a large factor in the cases tested as planning times were kept short (less than 1s per agent) compared to the size of the prediction horizon (around 10s). These times varied with the depth of the tree, so by adjusting the manoeuvre times and time horizons this approach can be flexible in its detail or reach. In addition, the control section would conform to the planned motion by the time the expected conflict was reached even if it started avoiding from a slightly different state than predicted.

Over the research work, differences between simulation and practical work did not introduce significant factors to be considered or designed around in our case. They could be negligible: with respect to the requirements and tolerances of our goals, due to the development tools used (e.g. real time sufficiently emulated by sim time in ROS time), or because the vehicle system is very kinematic. Similar considerations were either pragmatic or concerned with consistency between planning and execution, not simulation.

Overall, this research met its goal of implementing cooperative collision avoidance on the voyager UGV platform. Future work on the system would explore alternatives to improve performance, close gaps in assumptions like dynamic obstacle tracking, or look at practicalities like necessary network architectures.

References

1. J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, R. Mangharam, Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing, Intelligent Transportation Systems, **3**, 458-488 (2022)
2. A. AbdElmoniem, A. Osama, M. Abdelaziz, S. A. Maged, A path-tracking algorithm using predictive Stanley lateral controller, International Journal of Advanced Robotic Systems, 1-11, Dec (2020), **17(6)**, https://doi.org/10.1177/1729881420974852
3. M. Radmanesh, M. Kumar, P. H. Guentert, M. Sarim, Overview of path-planning and obstacle avoidance algorithms for UAVs: A comparative study, Unmanned Systems, **6**, 1–24 (2018), https://doi.org/10.1142/S2301385018400022
4. J. Barraquand and J. Latombe, Robot motion planning: A distributed representation approach, The International Journal of Robotics Research, **10(6)**, 628–649 (1991).
5. V. Roberge, M. Tarbouchi, G. Labonte, Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning, IEEE Transactions on Industrial Informatics, **9(1)**, 132–141 (2013).
6. V. R. Desaraju, J. P. How, Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees, 2011 IEEE International Conference on Robotics and Automation, 4956-4961 (2011), https://doi.org/10.1109/ICRA.2011.5980392

7. M. Hughes, Autonomous Guidance and Conflict Avoidance for Multiple Unmanned Aerial Vehicles (UAVs) in Urban Environments, Master Thesis, University of Stellenbosch, Faculty of Engineering (2023)
8. T. Moore and D. Stouch, A Generalized Extended Kalman Filter Implementation for the Robot Operating System, IAS-13, Jul (2014)
9. M. Labbé and F. Michaud, Multi-Session Visual SLAM for Illumination-Invariant Re-Localization in Indoor Environments, Frontiers in Robotics and AI, **9** (2022)
10. R. Boulic and O. Renault, 3D Hierarchies for Animation, New trends in animation and visualization, 59-77, Aug (Wiley-Interscience, 1991)