A Planning System for Autonomous Ground Vehicles Operating in Unstructured Dynamic Environments

Thomas Allen, James Underwood, Steven Scheding
ARC Centre of Excellence for Autonomous Systems (CAS),
Australian Centre for Field Robotics,
University of Sydney, NSW, Australia
{t.allen, j.underwood, scheding}@acfr.usyd.edu.au

Abstract

This paper describes the design and implementation of a path planning system for an autonomous ground vehicle. The system is designed to be flexible, allowing any planning algorithm to be used and any topology of data to be planned over. It employs a hierarchical separation of two planning modules in conjunction with a vehicle model, to achieve continued vehicle motion while planning and the ability to act as either a deliberative or reactive planner, or a hybrid of both types.

Results from both simulation and field trials are presented, and demonstrate the effectiveness of this architecture on a large outdoor ground vehicle. The contributions of this paper are twofold: a flexible planning system capable of large scale missions for autonomous vehicles; and the use of a vehicle model to determine the requirements for safe operation without slowing the vehicle, and the conditions under which this cannot be achieved.

1 Introduction

Path planning for Autonomous Ground Vehicle (AGV) applications most often occurs in dynamic environments, where the timeliness of goal acquisition is typically a high priority. Despite this, a significant proportion of implementations in the literature make the assumption of a static world while the vehicle is in motion, or stop the vehicle while a new plan is prepared. This paper presents a planning system which neither makes this assumption nor needs to stop while computing a new plan, allowing for long-range plans through dynamic environments to be achieved in a timely manner.

Other approaches in the literature can also achieve this objective, but were deemed unsuitable for the particular implementation described in this paper, since they lacked the ability to determine when they were incapable of planning safely. The main features of the system presented here are that it allows any planning algorithm to be used internally, and offers certain guarantees of vehicle safety if the plans can be computed sufficiently fast. This first feature means that the system can be implemented with equal performance to any other planning algorithm, by using this algorithm internally. The second feature means that the trade-off between the frequency of adjustments to the plan, and the distance along a plan to which a vehicle is committed can be optimised (this is discussed further in sections 3.1 and 3.5).

Given that most real-world scenarios have a vehicle's sensors build a model of the environment incrementally, even when the real world is static, this model is clearly dynamic. As a result, the optimum path to a goal should be re-evaluated as this information is updated. The literature on this problem typically falls into two main approaches: deliberative planners, which replan the entire path as fast as possible (or when the environment model changes); and reactive planners, which react to changes in the environment model by modifying the existing plan. Deliberative planning systems have the benefit that any planning algorithm can be used, whereas reactive planners are typically a self-contained algorithm, but can be hundreds of times faster as they need not replan the entire mission at each step [LaValle, 2006].

In the deliberative replanning category, Zelinsky [1992], Stentz [1994], and others describe a brute-force replanning approach, where a complete plan to the goal is replanned every time the perceived environment changes. Such an approach scales poorly with the distance to the goal, typically $O(n^2)$ for a search optimisation algorithm such as A^* [Hart *et al.*, 1968]. This approach can be improved by using more efficient data representations for cost, such as quad-trees [Zelinsky, 1992; Yahja *et al.*, 1998]. An alternative approach known as anytime planning makes a sub-optimal but feasible initial plan, and then iteratively improves this plan until time runs out [Zilberstein and Russell, 1995]. Typically the process is then repeated in a brute-force manner, but with the advantage that a plan can be aborted if the cost information changes significantly, and a new feasible plan made available rapidly. Likhachev *et al.* [2003; 2005] further describe a reactive form of the *Anytime* A^* algorithm with promising results.

Reactive planners tend to be more efficient than deliberative replanning methods, even with an optimised data representation and planner, but are typically more complicated to implement [LaValle, 2006]. The D^* algorithm, [Stentz, 1993; 1994; 1995; Stentz and Hebert, 1995] is arguably the best reactive planner for autonomous robotics applications, and operates by propagating changes in cost information throughout the state space of the planner. With these cost propagations in place, a new plan can be generated far more efficiently, as it can reuse the existing plan, accounting for the changed regions which comprise only a subset of the data. Variants of the D^* algorithm such as D^* *Lite* [Koenig and Likhachev, 2002] and *Field* D^* [Ferguson and Stentz, 2006] further improve the efficiency and applicability.

Both classes of planners have been shown to be effective in particular applications, and reactive planners using the D^* algorithm have been implemented in several AGV systems, such as the Mars Rovers [Tompkins *et al.*, 2004; Carsten *et al.*, 2007]. Despite the many successful implementations of such methods, the vast majority only work as well as described for the particular application they solve. Carsten *et al.* [2007] describe such a limitation with regards to large scale mission planning for the Mars Rovers, where the system was suitable for its original task, but ran into failures when this task was extended.

2 Motivation

The main motivation for the planning system described in this paper is the desire to achieve persistent autonomy, which requires a number of features from a planner that are unavailable in current implementations. This paper takes the definition of persistent autonomy as the ability for a vehicle to autonomously achieve goals in an unbounded area, and to determine the conditions under which this ability is compromised. A further motivation is that for many real-world AGV systems, including the Argo vehicle upon which this planning system is implemented, it is important to reach these goals promptly. This means that it is infeasible (or at least undesirable) to stop the vehicle to acquire additional sensor data or while a new plan is being prepared.

An effective path planning system should be reusable by different vehicles, in different environments, and with completely different path optimisation objectives. The system described in this paper achieves these requirements by designing each sub-section of the system in a generic and reusable manner. It separates the sensor data information from the cost information, allowing for different vehicles to plan differently through the same environment. It further separates this cost information from the data structure used to encapsulate it, meaning that an optimised data structure such as the framed quad-trees suggested by Yahja *et al.* [1998] can be used as easily as a simple grid. Finally, it provides a general solution to achieve the combined objectives of traversing an unknown environment along an optimal path, and avoiding obstacles while doing so.

As described in section 1, the assumption of a static environment is untenable for a real world system in which sensors produce a model of the world incrementally. Given that the environment must thus be considered dynamic, this motivates a system which commits to as little of a plan as possible, on the assumption that it may change in the next planning cycle (whether this is reactive or replanning). Many current planning systems which continue to execute while refining their plans – such as those using the D^* or anytime algorithms – solve this problem by allowing both the plans and the vehicle's execution of them to be imperfect, and then correct the errors due to this in the next planning cycle.

This solution is sub-optimal, because it results in errors from the first plan and execution propagating to the subsequent plans. Consider a vehicle with a constant cross-track error, which is slightly off its desired path. Each new plan will instruct this vehicle to move back onto the desired path, but the cross-track error will prevent this from occurring, instead moving the vehicle onwards with this same error. All subsequent plans will continue to request movement towards the desired path and yet never achieve it, propagating this error onwards. If the error is increasing, rather than constant, this problem is continually exacerbated.

The system presented in this paper instead predicts the future position of the vehicle after the estimated planning time for the next cycle, and plans from this point onwards. This vehicle model will necessarily be imprecise, and hence this solution will also be imperfect. However, because the vehicle model is incorporated in the planning system, this system can solve for problems such as the cross-track error example by feeding the error between predicted and measured vehicle behaviour back into the model and controllers.

A final motivating factor for the separation of the planning algorithm from the overall planning system, is that an algorithm can be optimised or replaced, yet the vehicle's behaviour remain constant. In the system presented here, the path plans can be performed by the A^* algorithm just as easily as the D^* algorithm, Vector Force Fields, or even Markov Decision Processes. Regardless of which is used, the type of behaviours exhibited by the vehicle are the same.

3 Design

The motivating factors described above require a general algorithm which can support any type of planner, and one which avoids the problems associated with planning while moving. This section first formulates the minimum reaction time required by the vehicle to avoid obstacles, and illustrates the trade off between this reaction time, and the planning horizon. Next, it presents a design for a replanning system which takes this reaction time into account, and achieves the twin objectives of unknown terrain traversal, and local obstacle avoidance.

3.1 Planning While Moving

All planning takes a finite time – regardless of the algorithm used – in which a vehicle may have moved a significant distance. As a result of this motion, planning systems which do not stop the vehicle while planning must invalidate the first section of a plan before it is sent to the vehicle, lest it turn around to reach the start of this new plan, which it has already passed. This solution is potentially hazardous if the environment changes significantly during the planning time, but is also inefficient because computation time is wasted on a segment of path which is then invalidated.

An alternative solution is to use a vehicle model to predict the future position of the vehicle at the time the plan is completed. The plan can then start from this future position, avoiding the need to invalidate any of the plan. Furthermore, this future position can be explicitly calculated to reflect the dynamics of the vehicle.

Let the vehicle's current velocity, $\dot{x_t}$, be a function of the control inputs, u_t, and the vehicle's current state, x_t:

$$\dot{x_t} = f(u_t, x_t) \tag{1}$$

The future state, $x_{t+\Delta t}$, can then be calculated as the integral of the velocity, over some time interval, Δt:

$$x_{t+\Delta t} = \int_t^{t+\Delta t} \dot{x_t} dt = \int_t^{t+\Delta t} f(u_t, x_t) dt \tag{2}$$

This integral can either be solved numerically, or approximately by a Taylor series, to give the future state of the vehicle for any time increment, Δt.

A number of practical features arise from equation 2. Since it is used to predict the future position of the vehicle after the time required for a plan, the error between this prediction and the real position is directly proportional to the planning time, t_P. Thus, for a fixed state error, a planning system can take a longer time to plan, at the expense of needing a better vehicle model or solution to equation 2. Alternatively, if the planning time is reduced then the quality of the model becomes less important, since the vehicle moves less far in each planning cycle. Dynamic events which cannot be predicted by the vehicle model can thus be reduced to an acceptable level if the planning time can likewise be reduced.

The most significant attribute of equation 2 is a guarantee of vehicle safety. Equation 2 can be used to determine the time, t_S, and distance, s_S, required to stop the vehicle for a given velocity, $\dot{x}$. If an obstacle is detected upon the vehicle's current path at a distance less than s_S, it is impossible to avoid it. Any obstacle further than this distance is safe, because the vehicle can stop before reaching it. To avoid an obstacle without the vehicle needing to reduce its velocity, the obstacle must be further than a distance of

$$s_P = t_P \times \dot{x} \tag{3}$$

where s_P is thus the maximum distance the vehicle can move at its current velocity, during the time taken to plan.

Figure 1 shows the required vehicle actions for obstacles within particular ranges of the vehicle. If

$$t_P \le t_S \tag{4}$$

then the area of region two reduces to zero, indicating that planning is sufficiently fast to produce plans which avoid all obstacles save those which are completely unavoidable given the vehicle's current velocity.

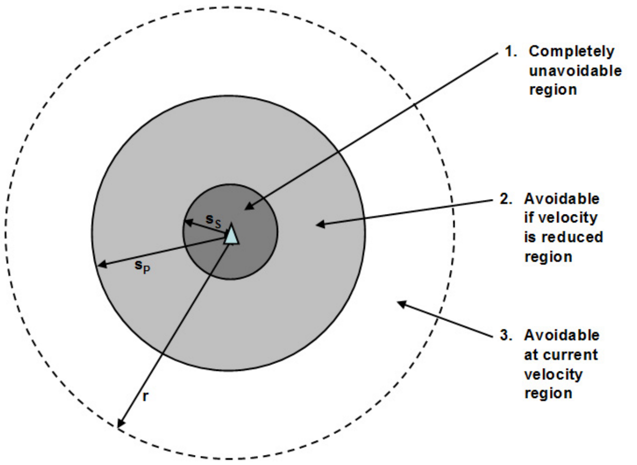
1. Completely unavoidable region
2. Avoidable if velocity is reduced region
3. Avoidable at current velocity region
s_S
s_P
r

Figure 1: Required vehicle actions for obstacles on the current path within certain ranges.

The following section describes a planning system which makes use of this inequality to guarantee obstacle avoidance within a local region around the vehicle.

3.2 An Abstract Planner Construct

To meet the requirements motivated in the preceding section, this planning system requires an abstract Planner construct to represent the means by which an optimum path is determined between two points. A Planner then uses several other constructs, designed in such a way that plans can be evaluated over any type and structure of data. This Planner construct is described as abstract because it does not implement a planning algorithm, but instead sets up a series of constructs with which such an algorithm can be implemented. The constructs required to define a Planner are as follows, and their inheritance is shown in the Unified Markup Language (UML) diagram in figure 2.

- A CostSet is a set of cost data, X_C, where each node, $x_i \in X_C$, represents the cost for a plan to pass through the location represented by i[1].
- A ConnectivityFunction defines the topology of a CostSet, X_C, allowing the data structure of the CostSet to be independent of this topology. It takes a given node, $x_i \in X_C$, and returns a list of all n nodes, $\{x_j, x_{j+1}, ..., x_{j+(n-1)}\}$, connected to it.
- A DataSet is a set of arbitrary data, Y_D, in a particular topology, where each node, $y_i \in Y_D$, is a single data point.
- A CostFunction is a function which returns the CostSet element, x_i, whose cost is calculated by applying a function to those nodes in the DataSet, Y_D, which contribute to this element.
- A DataCostFunctionCostSet is a special type of CostSet, X_C, which uses a CostFunction to produce the element, $x_i \in X_C$, whenever it is accessed.
- Finally, a Planner takes a starting node, x_S, and a goal node, x_G, in a CostSet, X_C, and returns the lowest cost path between them as a connected set of nodes, $X_P = \{x_S, ..., x_G\}$.

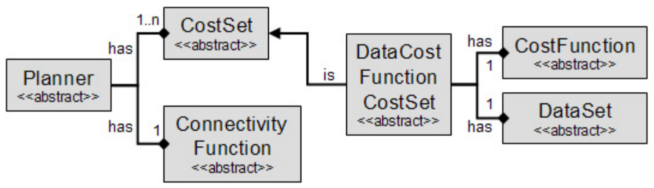

Figure 2: UML Diagram of the Planner construct.

This structure ensures that the data required for a Planner can have any type and structure, provided that the implementation can define a CostSet and ConnectivityFunction appropriately. Where a CostSet is not trivially described by the data itself, a DataCostFunctionCostSet can be used to build a CostSet from a DataSet and a CostFunction. For example, if a vehicle's sensors determine the height of an environment in 1cm squares and store these in a DataSet, a DataCostFunctionCostSet could produce a grid-based CostSet with larger cells by using a CostFunction to take the average height from the DataSet in the area represented by each CostSet cell.

As figure 2 shows, a Planner has a ConnectivityFunction, and also n CostSets. Regardless of how the Planner is implemented, provided that it returns the lowest cost path using this ConnectivityFunction over the sum of the CostSets[2], it can be treated as an independent module and be used in a path planning system as described below.

3.3 Planning in the Local Sensor Region

For typical AGV operation in dynamic environments, new environmental information will be restricted to the region around the vehicle which its sensors can perceive, or a larger region if additional data is being fused from other sources. This sensor region is typically much smaller than the overall area in which the vehicle operates, and can thus be planned over in a significantly faster time.

From figure 1, if this sensor region covers all locations the vehicle can reach within its planning time, that is,

$$r \geq s_P \tag{5}$$

and equation 4 also holds, then planning over this region is sufficient to achieve local obstacle avoidance without needing to slow the vehicle while planning.

3.4 The Replanner System

The design of a Replanner system is now presented, which makes use of the Planner construct described above. Figure 3 shows a UML diagram of this system, and a pseudo-code representation of the algorithm is shown in figure 4.

A Replanner has two Planners, known as the LocalPlanner, and GlobalPlanner, and also asynchronously receives updates to its stored CostMap(s) and vehicle state. It takes a set of n mission goals, $X_G = \{x_1, ..., x_n\}$, of which nodes x_i and x_{i+1} are sent to the GlobalPlanner as the start and goal nodes to plan between, whenever node x_i has been reached by the vehicle. If the global plan is ever invalidated (as described below), the starting node is instead given as x_V, the current vehicle position. Whenever a global plan is valid, the LocalPlanner is run as often as possible. Ideally, this should be as fast or faster than new cost data is received, so that each plan is still optimal with respect to the current cost data at

[1]This representation differs from the typical cost graph, where costs are associated with the connections between nodes, rather than the nodes themselves. A CostSet is equally general, and may be a cost graph, or any other topology of information.

[2]Where the overall cost cannot be expressed as a weighted sum of the costs from several CostSets, it can instead be implemented as a single DataCostFunctionCostSet, using an appropriate CostFunction and DataSet which combine the data from all the sources.

```
Main Loop
    while( $missionGoals \neq \emptyset$ )
        get $currentVehiclePosition$
        if( $currentVehiclePosition$ is within tolerance of $currentMissionGoal$ )
            remove $currentMissionGoal$ from $missionGoals$
        if( $globalPath = \emptyset$ )
            find $globalPath$ from $currentVehiclePosition$ to $currentMissionGoal$
            if( successful )
                $vehiclePath = globalPath$
            else
                return FAILURE
        else
            if( $dataStore$ has changed, or $currentVehiclePosition$ has changed )
                get $predictedVehiclePosition$ after $averageLocalPlanningTime$
                set $localPlanningStart$ to the closest point in $globalPath$ after $predictedVehiclePosition$
                set $localPlanningGoal$ to the closest point in $globalPath$ before $localPlanRange$
                find $localPath$ from $localPlanningStart$ to $localPlanningGoal$
                adjust $averageLocalPlanningTime$ by $measuredLocalPlanningTime$
                if( successful )
                    $vehiclePath = localPath$
                else
                    clear $vehiclePath$, stopping vehicle
                    $globalPath = \emptyset$

Update($CostInformation$)
    update $dataStore$ with $CostInformation$

Update($NavigationInformation$)
    update $currentVehiclePosition$ with $NavigationInformation$
```

Figure 4: Pseudo-code presentation of the of the complete Replanner system. The main loop and the update functions run asynchronously (with appropriate data locking around the cost and navigation information).

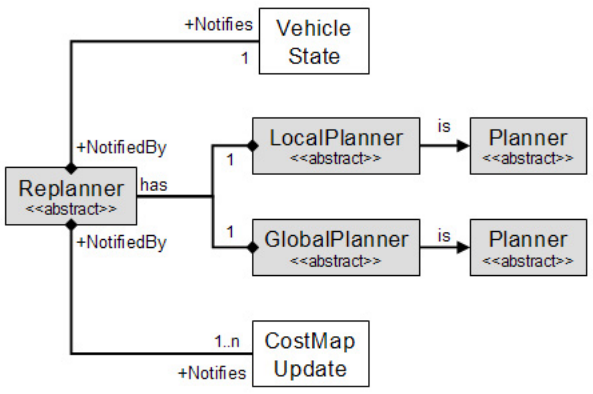

Figure 3: UML Diagram of the Replanner construct.

the time it is completed. A local plan can fail if either the goal point lies behind an impassable obstacle (given the current cost data), or because the Planner it uses will abort a plan upon some condition, such as a time-limit. If this occurs, the vehicle is safely brought to a stop and the global plan is invalidated. If equation 4 holds, this is the only situation in which the Replanning system need ever stop the vehicle while planning.

The Replanner uses the vehicle dynamics model described above to predict the vehicle's position once the next local plan has been computed, using an estimate of this planning time (such as a rolling average of how long previous local plans have taken). The LocalPlanner is then given a starting node that is the next point on the global plan from this estimated vehicle position. The goal node is the furthest point on the global plan that is closer than a specified local planning distance, and which is not an obstacle. This choice of starting and goal nodes is made such that the LocalPlanner is always following the global plan, but locally adjusting it as new information is received. Since the starting position reflects the dynamics of the vehicle, this method explicitly allows the vehicle to continue moving while planning, and avoids the need to invalidate any parts of the plan.

The design presented here is more generic than the implementation described below, as it allows for the LocalPlanner and GlobalPlanner to be separate Planner constructs. However, since the Replanner is guaranteed to never be using both the LocalPlanner and GlobalPlanner simultaneously, it is possible for both of these to use the same Planner and CostSet, as in the implementation on the Argo vehicle. Alternatively, this design allows for the GlobalPlanner to be a much coarser data representation, or even use entirely different data (for example, using a satellite map of an area while the LocalPlanner uses onboard sensors).

3.5 Trade-Offs

As described previously, one of the main benefits of this Replanner design is the ability to determine the effects of adjusting parameters in the system. Defining the frequency at which the cost maps can be updated as f_C, and the frequency at which planning can be achieved as f_P, it is desirable that

$$f_P \geq f_C \tag{6}$$

in order that the system does not compute a plan that is already sub-optimal with respect to the latest cost information by the time it is completed. Equally, it is desirable that equation 4 is upheld, to ensure that all avoidable obstacles can be planned around without the need to slow the vehicle. Equation 4 can also be expressed in terms of distances:

$$s_P \leq s_S \tag{7}$$

The stopping distance, s_S, is a function of the vehicle velocity, $\dot{x}$, and from equation 3, the planning distance, s_P, is likewise. As a result, it is clear that adjusting the vehicle velocity can have no effect on the inequality of equations 4 or 7.

The only other variable that can be adjusted to achieve equation 4 is the planning time, t_P, and clearly

$$t_P = \frac{1}{f_P} \tag{8}$$

by definition. This planning frequency is a function of the computational speed of the planning algorithm, the cost data resolution, the length of the local plan, and potentially other factors. The planning algorithm cannot be optimised past a certain point, so this factor cannot be adjusted. The resolution of the cost data can be reduced to increase the planning frequency, but at a cost to the quality of the plans. Additionally, many implementations would be able to produce cost maps at a faster rate if the data resolution was reduced, meaning that this parameter may lie on both sides of the inequality in equation 6. This leaves the length of the local plan as the only parameter which can directly contribute to the the trade-offs described by equations 4 and 6.

Adjusting this parameter allows the Replanner system to operate anywhere along the scale between reactive and deliberative planners. If sufficient computational time is available, then the local plan length can be the entire length of the global plan, causing the system to act like a brute-force deliberative planner. Alternatively, if the minimum local plan length is just greater than the stopping distance (so that the vehicle is never able to drive past the end of a plan into potentially hazardous areas), the system behaves like a purely reactive planner.

Since the local plans find a path to the furthest point along the global plan allowed by their maximum length, the local plans are better if this parameter is longer, since they can take paths past larger regions of the now sub-optimal global plan, as shown in figure 5. The competing desires to see the local plans both as long and as frequent as possible allows this parameter to be optimised precisely. For optimal planning performance, the local plan length should be the largest distance possible to still allow for equation 6 to hold. The implementation of the Replanner system upon the Argo vehicle sets the local plan length in this manner, and is described in the following section.

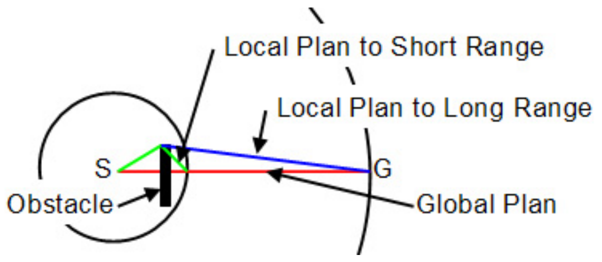

Figure 5: Local plans around an obstacle for a short maximum plan length (green), and a longer range (blue), illustrating that the longer the local plan length, the more direct the planned route.

4 Implementation

The Replanner design was written in a C++ framework, and a particular implementation was then created for the Argo vehicle, shown in figure 6. C++ was chosen primarily because the Planner and Replanner constructs can be easily represented as abstract classes in a object-oriented language. Implementing a specific instance of a Replanner is then a matter of inheriting from these abstract base classes and following the required interfaces. For Argo vehicle application, the A^* algorithm was used to implement a Planner, and was chosen for the simplicity of its implementation, and the provable optimality of its plans (with respect to the cost data) [Pearl, 1984]. A two-dimensional grid topology, implemented as a quadtree, was used for the CostSet, since the current requirements of the Argo vehicle do not require a more versatile representation.

Figure 7 illustrates the overall implementation of the A^* algorithm as a Planner, showing the abstract classes

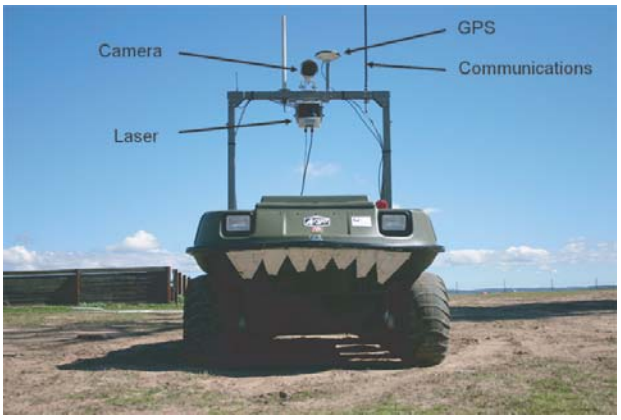

Figure 6: The Argo vehicle, showing sensors and communications hardware.

in light grey, and the specific implementations in dark grey. This figure shows that the implementation uses a two-dimensional grid for a DataSet, which itself uses an efficient C++ quad-tree implementation so that uninitialised data points require no memory for storage. The cost information is constructed from this grid using a CostFunction which multiplies the data value of each node (described below) by the Euclidean distance to the goal node, with an adjustable weighting. The CostSet built from this data has each node connected by an eight-way grid function, (allowing a range of moves equivalent to a king in a game of chess).

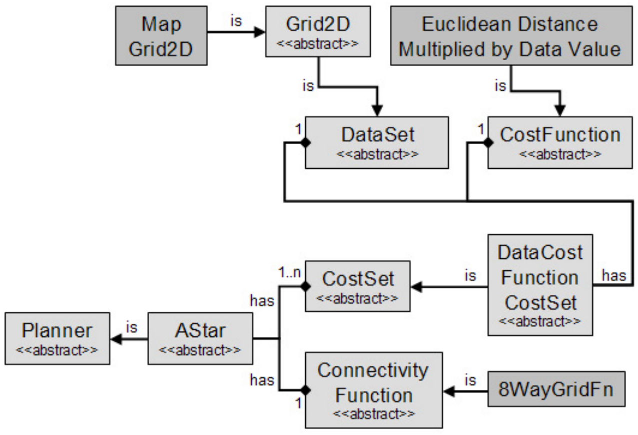

Figure 7: UML diagram showing the overall implementation of the A^* algorithm as a Planner.

As implemented upon the Argo vehicle, the DataSet is built incrementally from the laser data points acquired by the vehicle's sensors. This is built into a terrain cost grid by assessing areas which have either steep gradients between neighbouring grid cells, or a large range of heights within a single cell. These thresholds for terrain steepness and untraversable heights depend on the particular vehicle used; for a different class of vehicle an entirely different method may be more appropriate. Each cost cell in the grid is then expanded by an amount proportional to the vehicle's dimensions, in a process functionally equivalent to a Configuration-Space transform, [Russell and Norvig, 2003; LaValle, 2006].

The Planner implementation allows for any of the n CostSets to veto a node, marking it as untraversable. For example, the Argo vehicle's implementation uses this feature to mark the boundaries of the testing facility, which are otherwise undetectable by the vehicle's sensors. This process is equivalent to making the cost prohibitive, but the A^* algorithm implementation then completely removes this node from the search space, which is a more efficient solution as it reduces the search time by focussing only within the traversable region. The final output of the Planner is an ordered set of nodes, connected by the moves allowed by the ConnectivityFunction.

The Argo vehicle carries three separate computers to manage vehicle control, data processing, and the replanning system, each of which comprises a PC104 stack with a single-core Pentium III processor. With a grid cell size of 30cm, local plans to 11m ahead of the vehicle (the point where the laser scanner hits the ground) can be performed at greater than 20Hz. This is faster than the vehicle can generate an updated CostMap of this region, and far faster than required for equation 4 to hold true. As a result, this implementation is sufficiently real-time for the current requirements of the Argo vehicle.

Finally, this implementation has a number of variable parameters (separate to those in the Replanner system, described in section 3.5) which directly affect the behaviour of the system and the vehicle. Uninitialised data in the CostMap is given a fixed cost which affects the vehicle's tendency to plan through unknown terrain. If this cost is low, then the system will preference paths through unknown areas rather than paths over costly terrain, and vice versa. To allow multiple CostSets to be planned over simultaneously, the Planner construct assumes that their costs can be added in a weighted sum (however, if this is not possible, they can be combined into a single DataCostFunctionCostSet in whatever combination is appropriate). The weighting of each CostSet in this sum can thus be adjusted to reflect the importance of avoiding areas of high cost in each set with respect to the others. The last main parameter is the weighting of plan length versus cost, resulting in a preference for longer paths which travel over less costly terrain if it is raised, and vice versa if it is lowered.

5 Results

Results of this Replanner implementation are now presented, firstly in simulation, and secondly running in real-time on the vehicle, with no changes to the implementation. The simulated results used an accurate vehicle model and simulated drive-train, and read in cost information from a pre-generated image in an area around the vehicle equivalent to the sensor field of view, at the same rate as a CostMap could be generated by the vehicle. The simulation was run both with cost information obtained from a helicopter fly-over of the test area, where the helicopter carried the same SICK laser scanner as the Argo vehicle, and also with data directly from the vehicle driving through the same area. The simulated vehicle exhibited similar behaviour with both sources of cost information. The real-time results used exactly the same implementation, save that the CostMap was generated in real-time from the vehicle's laser scanner.

Figure 8 shows a time sequence of the Replanner system in operation, using the simulated Argo vehicle, and pre-generated cost information. In the figures, the background image is a high resolution photo of the test area, and the cost information is incrementally overlayed upon it as the vehicle moves. The pink circle shows the current mission goal, and the red circles show the current global plan to this goal. The green circles are the current local plan, and the blue circles are the path to which the vehicle is committed, (which in these images is a fixed length because it only reflects the vehicle's maximum stopping distance, since the planning time was constant).

Figure 8-1 shows the initial global plan, which in the absence of cost data, is merely a straight line to the goal. With no obstacles seen in the local region, the green local plan simply follows the global plan. Figure 8-2 shows the green local plan diverging from the global plan to take a better route, given the new information just acquired by the sensors. This frame also shows that the global plan crosses an obstacle which has just been discovered by the vehicle's sensors. This has invalidated the global plan, stopping the vehicle (hence the blue committed path is not seen). Figure 8-3 then shows the new global plan which avoids this obstacle, from the vehicle's current position to the goal. Finally, figure 8-4 shows the vehicle's path with small red circles, and the total cost information gathered by the vehicle during this mission.

The attached video [3] shows the same Replanner implementation, running in real-time on the Argo vehicle. The vehicle was placed in an area of the test facility with no initial environment information, and was instructed to drive approximately 150m to the far side of a thicket of dense scrub. The video shows the vehicle performing this mission, illustrating the vehicle needing to redo its global plan and back-track at one point. A screen capture of the user interface software illustrates its route midway through the mission. The same mission (given by three waypoints) was then repeated using the helicopter fly-over as initial cost data. With this initial data, the vehicle calculated a far better path (with respect to its CostFunction), first driving away from the goal to reach a flat road, then proceeding around the entire scrub area. The last screen capture demonstrates the faster and less complicated path performed in this second mission. In both cases the vehicle reached the goal in an appropriate manner, with the Replanner system operating as expected. Subsequent missions with improved communications hardware on the Argo vehicle have used the same Replanner system over distances of around 2km with equal success.

[3] Available at `http://www-personal.acfr.usyd.edu.au/t.allen/ACRA2007/ReplannerResults.mpg`

6 Discussion and Future Work

One criticism of this Replanner approach is that predicting the future position of the vehicle and planning from this point is functionally equivalent to a reactive planner which allows imperfect execution of its plans, and then corrects for this in the next cycle. This is correct, but neglects the advantages of the Replanner system, in that its makes the trade-offs involved explicit and adjustable. A reactive planner such as the D^* algorithm is forced to account for all updated cost data at each planning cycle, whereas the Replanner approach can potentially plan over only a smaller subsection of this changed data. Even if the reactive planner can plan through this updated data in an efficient manner, depending on the quantity of data involved, this can still be a computationally expensive process. An anytime algorithm can partially solve this problem by aborting the plans earlier, but sacrifices their optimality to do so. By comparison, the Replanner approach can continue to produce optimal local plans (with respect to the current cost data) at the required rate, and the only trade-off is to limit how much they can improve upon the global plan because they must follow it more closely.

This Replanner system has been shown to be practical and efficient in its implementation on the Argo vehicle, yet there are several areas in which improvements could be made, and where further research is likely. Even with a planning system which generates the optimum path with respect to the cost information, it is still difficult to accurately control a vehicle along it. There are many solutions to this problem, such as using a non-holonomic planner which can reject paths that are impossible for the vehicle to follow, or applying a path optimisation stage to the algorithm to smooth or otherwise modify the path. The state lattices proposed by Kelly *et al.* [Pivtoraiko and Kelly, 2005; Knepper and Kelly, 2006] are an effective such method, and can be readily implemented in the Replanner framework. For the skid-steered Argo vehicle, a wheel-slip estimator has been designed, which should allow the vehicle to better control its steering at all speeds, resulting in better path following.

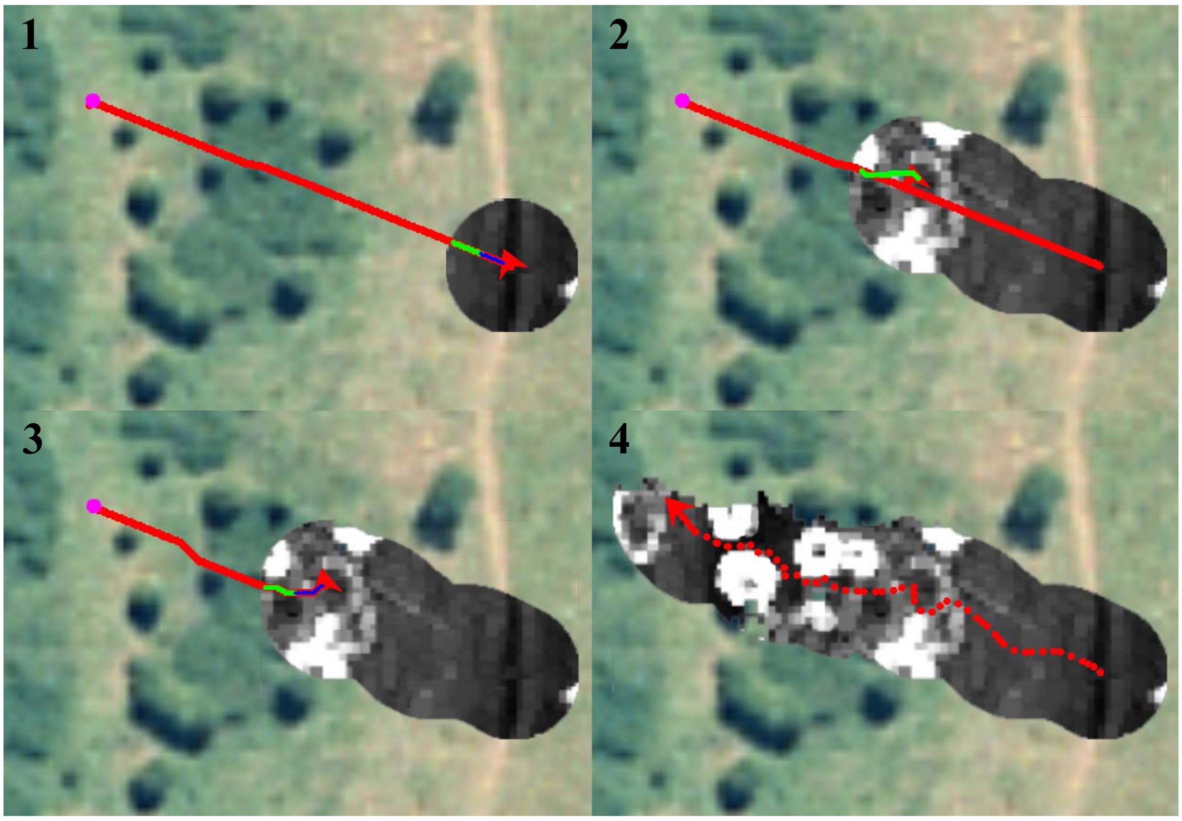

Figure 8: The Replanner algorithm in simulation.

All discussion in this paper has described the paths produced as optimal with respect to the cost information, and not necessarily optimal if this information is wrong or insufficient. It is arguable however that if the cost information is deficient, then subsequent problems are the result of this, and not the planning system used. The Replanner system presented here removes many of the difficulties associated with planning, while moving them to the area of cost map generation and vehicle control. In the Argo implementation, these cost maps are able to be generated in real-time at approximately 10Hz, sufficiently accurately to allow for optimal planning. If this was not the case, then the results obtained – while still correct with respect to the cost information – would not be as satisfactory as presented here. For the Argo implementation, the largest computation effort for cost map generation is expanding the obstacles by the vehicle's dimensions. It is believed that this task could be equivalently performed using an efficient image dilation technique directly on a consumer Graphical Processing Unit (GPU), which would free up CPU time and allow the vehicle to plan even faster.

Future work in this area will focus on two main objectives: the fusion of additional data from other vehicles and the subsequent use of the Replanner system for multiple vehicle control; and testing the system with truly dynamic obstacles such as other vehicles, livestock, or pedestrians. It is believed that the Replanner system is general enough to handle such situations, but only if the cost information can represent the changing environment sufficiently quickly and accurately, and the vehicle can be controlled effectively along the rapidly changing paths.

7 Conclusion

This paper has presented the Replanner system for AGV navigation in unstructured dynamic environments. The system has the ability to operate anywhere along the scale between reactive and deliberative planners, and the optimum point on this scale can be determined for any given implementation. Results of this system as implemented on a large ground vehicle known as the Argo were presented, and show that the system is suitable for achieving persistent autonomy. Future work will focus on producing cost maps in a manner to allow dynamic obstacle avoidance, and the use of the system for multiple vehicle control.

The main contributions of this paper are twofold: a framework for a flexible planning system to achieve persistent autonomy, capable of using any planning algorithm, and any type and structure of cost data; and the use of a vehicle model in the planning system, which allows the requirements for safe operation to be determined, and without the need to slow or stop the vehicle while planning.

References

[Carsten *et al.*, 2007] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz. Global path planning on board the mars exploration rovers. In *IEEE Aerospace Conference*, 2007.

[Ferguson and Stentz, 2006] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field d* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2 2006.

[Hart *et al.*, 1968] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 2:100–107, 1968.

[Knepper and Kelly, 2006] R. A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[Koenig and Likhachev, 2002] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *International Conference on Robotics and Automation*, 2002.

[LaValle, 2006] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Available from `http://planning.cs.uiuc.edu/`.

[Likhachev *et al.*, 2003] M. Likhachev, G. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Neural Information Processing Systems*, 2003.

[Likhachev *et al.*, 2005] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *International Conference on Automated Planning and Scheduling*, 2005.

[Pearl, 1984] J. Pearl. *Heuristics.* Addison-Wesley, 1984.

[Pivtoraiko and Kelly, 2005] M. Pivtoraiko and A. Kelly. Efficient constrained path planning via search state lattices. In *The 8th International Symposium on Artifical Intelligence, Robotics, and Automation in Space*, 2005.

[Russell and Norvig, 2003] S. Russell and P. Norvig. *Artifical Intelligence: A Modern Approach.* Prentice Hall, 2 edition, 2003.

[Stentz and Hebert, 1995] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. In *International Conference on Intelligent Robots and Systems*, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1995.

[Stentz, 1993] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical report CMU-RI-TR-93-20, Carnegie Mellon Robotics Institute, 1993.

[Stentz, 1994] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation*, pages 3310–3317, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1994.

[Stentz, 1995] A. Stentz. The focussed d* algorithm for real-time replanning. In *International Joint Conferences on Artificial Intelligence*, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1995.

[Tompkins *et al.*, 2004] P. Tompkins, A. Stentz, and D. Wettergreen. Global path planning for mars rover exploration. In *IEEE Aerospace Conference*, 2004.

[Yahja *et al.*, 1998] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *IEEE Conference on Robotics and Automation*, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1998.

[Zelinsky, 1992] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6), 12 1992.

[Zilberstein and Russell, 1995] S. Zilberstein and S. Russell. *Imprecise and Approximate Computation*, chapter Approximate Reasoning using Anytime Algorithms. Kluwer Academic Publishers, 1995.