Continuous Curvature Trajectory Generation with Obstacle Avoidance for Car-Like Robots

Simon Thompson and Satoshi Kagami
Digital Human Research Center
National Institute of Advanced Industrial Science and Technology
Aomi, Koto-ku, Japan
Email: simon.thompson@aist.go.jp

Abstract

This paper presents an extension of cubic curvature polynomial trajectory planning to include a mechanism for obstacle avoidance. Cubic polynomials have been used to describe curvature continuous trajectories for car like robots. From known start and end robot postures, (position, orientation and curvature) a continuous trajectory can be decided. Here we extend cubic polynomial trajectories to fourth order polynomials, and introduce a cost function, describing accumulated distance to obstacles along a trajectory, to the robot posture vector. Such trajectories, generated by a gradient descent method, satisfy continuity constraints and avoid obstacles. The method is implemented on a mobile robot system and experiments in real time trajectory planning and execution are conducted.

1. Introduction

Please follow the steps outlined below when submitting your manuscript to the IEEE Computer Society Press. Note there have been some changes to the measurements from previous instructions.

2. Introduction

Car-like mobile robots have limitations in their steering mechanisms which complicate trajectory generation. In order to achieve changes in orientation, such robots must translate while changing their steering angle. In addition, changes in linear position and steering angle should be continuous to reduce strain on the mechanical drive components and also to avoid the tracking errors which arise when following discontinuous trajectories. [1] proposed a trajectory generator which ensures continuous trajectories by modeling robot posture as a 4-tuple (x, y, θ, κ) (robot position, orientation ion and curvature) and describing the trajectory with a cubic curvature polynomial. Within a limited local region this approach can generate trajectories between arbitrary postures which are also continuous in curvature.

Robots however must operate in environments which may contain obstacles. The above approach does not take into account the possible presence of obstacles. In addition, if a proposed trajectory is subsequently found to produce a collision, there is no mechanism to revise the trajectory or to generate other possible trajectories other than through subdivision of the trajectory. This paper extends the cubic curvature polynomial trajectory generator to a fourth order curvature polynomial and generates continuous curvature paths which successfully avoid obstacles in the environment.

2.1. Search Based Path Planning

Very fast path planning algorithms exist for 2 dimensional search spaces [2]. These algorithms produce optimal paths to a goal state, but discretise the search space and largely ignore the steering dynamics of the physical systems. The generated path must be smoothed in order for a robot to execute it, and the smoothed path must also avoid obstacles. Brute force searching, however, for any space greater than 2 dimensions rapidly becomes computationally problematic, even when an initial optimal lower dimensional path is known. The cost of the high dimensional search can be lowered by reducing the size of the region the search is performed in. Even in small search spaces, path planning solutions can encounter discretisation problems when planning in confined spaces.

[3] combines 2 dimensional path planning with higher dimensional planning, performing a local search in five dimensional space once an optimal 2D path has been found. This limited search is still relatively expensive and is of course, over a discretised space.

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

In order to avoid discretisation errors, a geometric solution to the problem of planning local continuous trajectories is desirable. A good review of geometric motion planning is found in [4] One example of a geometric solution is that of [1], where cubic curvature polynomials are used to describe continuous local trajectories.

2.2. Cubic Curvature Polynomials

Cubic curvature polynomials describe trajectories which are continuous in curvature and therefore in steering controls. Although actual robot controls are typically in the form of velocities (linear and angular) specific trajectory shapes can be traversed with a range of velocities and so it makes sense to model trajectories as a function of distance rather than time.

From [1], the state equations for robot posture can be expressed as:

$$\dot{x} = V(t)cos(\theta(t)), \quad \dot{y} = V(t)sin(\theta(t))$$

$$\dot{\theta} = \kappa(t)V(t), \quad \dot{\kappa} = \dot{\alpha}(t)/L$$

where V is the linear velocity, α the angular velocity, t time, and L the wheel base. Modeling curvature as a cubic polynomial function of arc length s the system can be written:

$$\kappa(s) = \kappa_0 + as + bs^2 + cs^3 \tag{1}$$

$$\theta(s) = \theta_0 + \int_0^s \kappa(s) \tag{2}$$

$$x(s) = x_0 + \int_0^s cos(\theta(s)) \tag{3}$$

$$y(s) = y_0 + \int_0^s sin(\theta(s)) \tag{4}$$

By treating the problem as an inverse kinematics problem the co-efficients of the cubic polynomial and the arc length of the trajectory can be solved by iterative finite approximation of the Jacobian. This method generates continuous curvature trajectories for arbitrary start and finish postures within a local relative end posture envelope [1] of position between $-1.0m < y_T < 1.0m$ and $1.0m < x_T < 5.0m$, orientation $-4\pi/5 < \theta_T < 4\pi/5$, and curvature $-0.1/m < \kappa_T < 0.1/m$.

[1] note that cubic polynomials impose higher order constraints, ensuring smoothness in steering angle acceleration, than those imposed by earlier work. Also the method of generating cubic trajectories is more computational efficient than work with similar families of curves such as intrinsic splines[5].

2.3. Obstacle Avoidance

Cubic curvature trajectory planning does not take into account the presence or absence of obstacles along the trajectory route. Typically obstacle avoidance is achieved by recursively sub-dividing the trajectory when a possible collision with obstacles is detected. In this way, a spline of of a number of cubic trajectories can be constructed which will maintain continuous curvature, avoid obstacles and still reach the goal state. Trajectory generation in a new sub-division does not even take into account the obstacle whose existence necessitated the sub-division. In environments which contain narrow passages this method can require the generation of many sub-trajectories. Search based path planning of course allows for obstacle avoidance by the introduction of cost functions for node transitions [2][6], although discretisation problems can occur in narrow passages. In addition post search smoothing must usually be performed on the path.

Potential field methods [7] and the Dynamic Window Approach (DWA) [8] are local obstacle avoidance methods which seek to determine the current control based on a reactive sensor based approach. The evaluation functions however, only evaluate the next control, or if they do look ahead do not plan in velocity or curvature space.

Another approach to obstacle avoidance is the elastic-band approach[9] [10]. This approach uses bubbles attached to an elastic band representing the free space and the robot path, with bubbles pushing the path away from near obstacles. This approach can be computationally expensive when forming the initial path, although tracking and deformation of a local path can be achieved in real time. [11] notes that such approaches do not strictly meet non-holonomic constraints, and proposes a method which works on deforming a non-holonomic path defined by velocity inputs by perturbing the inputs themselves. The inputs are perturbed according to the gradient of the obstacle potential field surrounding the robot, and minimise this field by adjusting the input velocities.

In this research we propose to generate a local continuous curvature trajectory using geometric techniques which can incorporate some knowledge of close obstacles. Specifically we extend the cubic curvature polynomial formulation described by [1] to a fourth order polynomial and include a distance to obstacle cost term in the state description. Estimation of the polynomial then seeks to reach the target state with a cost of zero. This is similar to the work of [11], except that the current study deals directly with curvature, and deformation of the fourth order polynomial directly considers the boundary conditions (start and goal postures) as well as the curvature constraints. With this approach the discretisation problems associated with search based path planning are avoided, while trajectories

0-7695-2504-0/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

are planned taking account of close obstacles.

3. Problem Formulation

Continuous curvature trajectory planning in an environment containing obstacles can be formulated by augmenting the robot posture with a cost term, reflecting the cost of traveling along the current trajectory. In this case the [1]'s 4-tuple describing robot posture now becomes a 5-tuple: $(,x,y,\theta,\kappa,L)$, where L is the cost term. In this work, L is an inverse distance function which evaluates a given position x,y based on the inverse Euclidean distance to an obstacle. In adding a term to the state space, a control point or parameter must be added to the curvature polynomial in order to control the entire end point state. By adding a fourth order term to the curvature polynomial, we arrive at the following equations for the forward solution of trajectories:

$$\kappa(s) = \kappa_0 + as + bs^2 + cs^3 + ds^4 \tag{5}$$

$$\theta(s) = \theta_0 + \int_0^s \kappa(s) = \theta_0 + \kappa_0 s + \frac{as^2}{2} + \frac{bs^3}{3} + \frac{cs^4}{4} + \frac{ds^5}{5} \tag{6}$$

Equations 3 and 4 describing $x(s)$ and $y(s)$ remain the same, now integrating $\theta(s)$ as calculated in Equation 6. The new term L, describing the cost posed by the presence of obstacles is formulated as:

$$L(s) = \int_0^s \left(\frac{\lambda}{v_0} + \ldots + \frac{\lambda}{v_{N-1}}\right) \tag{7}$$

where λ is a constant describing the repulsive quality of obstacles and v_i is the distance between a robot posture and obstacle i. The posture $\underline{x}_0 = (x_0, y_0, \theta_0, \kappa_0, L_0)$ is the start posture while the target posture is $\underline{x}_T = (x_T, y_T, \theta_T, \kappa_T, \text{Ł}_T)$.

Like [1] we interpret the above state equations as a set of non-linear equations which can be solved for the parameters of the polynomial. In vector form the augmented state space $\underline{x} = [x, y, \theta, \kappa, L]^T$ can be expressed as a function of the fourth order polynomial parameters $\underline{p} = [a, b, c, d, s]^T$ as $\underline{x} = f(\underline{p})$. Again, as in [1] the system is linearised:

$$\Delta\underline{x} = [\frac{\partial \underline{x}}{\partial \underline{p}}]\Delta\underline{p} \tag{8}$$

and the inverse calculated as:

$$\Delta\underline{p} = [\frac{\partial \underline{x}}{\partial \underline{p}}]^{-1}\Delta\underline{x} \tag{9}$$

Using the inverse Jacobian the solution for $\underline{p}$ can be solved iteratively. In this paper we take the output of the cubic curvature polynomial trajectory generator [1] to be the initial guess and iteratively revise the parameters to avoid any obstacles in the local environment.

3.1. Obstacles

Obstacles are considered in trajectory generation by the addition of cost term L to the state space (x,y,θ,κ,L). In particular, $L(s)$ contains an integral of the sum of the inverse distance to obstacles as shown in Equation 7. Obstacles are modeled as point obstacles and the distance of obstacles from a position $(x(s),y(s))$ is given by Euclidean distance. The integral term of Equation 7 can then be stated as a sum over N obstacles:

$$L(s) = \int_0^s \Sigma_{i=0}^{N-1} \left(\frac{\lambda}{\sqrt{(x(s)-x_i)^2 + (y(s)-y_i)^2}}\right) \tag{10}$$

The definition of obstacles could be easily extended to include circles, lines and convex polygons. A more computationally expensive alternative would be to replace the explicit obstacle terms with a function which evaluates the current minimum distance to obstacles in a grid map.

3.2. Initialisation

Initialisation of the polynomial curve parameters for $\underline{p}$ is crucial for the iterative approximation to converge on a correct solution. Here, the output of a cubic curvature polynomial trajectory generator [1] is used to initialise the parameters a, b, c and s. The fourth order term d is simply initialised to 0.

3.3. Forward Solution

The forward solution of calculating a robot posture for a given trajectory is given by evaluating the system state $\underline{x} = f(\underline{p})$ for x, y, θ, κ, L for any value s. To solve the sine and cosine integrals involved in calculating $x(s)$ and $y(s)$ we use a numerical integration method, specifically Gauss-Legendre quadrature.

To assure the convergence on an acceptable solution, the cost function is rewritten to produce a minimum of zero for all acceptable trajectories. Acceptable trajectories are those which do not pass within a clearance distance D_C of any obstacles. The cost function thus becomes:

$$L = \left(\int_0^s \left(\Sigma_{i=0}^{N-1}\left(\frac{\lambda}{D(s)_i}\right)\right) - s\frac{N\lambda}{D_C}\right) \tag{11}$$

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

where the value of $D(s)_i$, the distance, of the robot along the trajectory at length s, to an obstacle i, if initialised to

$$D(s)_i = \sqrt{(x(s) - x_i)^2 + (y(s) - y_i)^2}$$

is:

$$D(s)_i = \begin{cases} D(s)_i & \text{if } D(s)_i < D_C; \\ D_C & \text{if } D(s)_i \geq D_C. \end{cases} \quad (12)$$

Substituting Equations 3, 4 and 6 the cost function can be directly solved using numeric quadrature. To solve $x(s)$ and $y(s)$ we apply Gauss-Legrendre with four evaluation points with sufficient results. However for $L(s)$, for $N > 1$, Gauss quadrature techniques prove unsuitable with a low number of evaluation points due to their nature of function approximation. To overcome this problem, the integral is approximated using the extended Trapezoidal method using a high number of function evaluation points (100). With this formulation, the forward solution of L will approach zero as the trajectory described by $\underline{p}$ approaches an acceptable trajectory.

3.4. Inverse Solution

The inverse solution takes two defined robot postures and finds an acceptable trajectory which will enable the robot to move from one posture to the other. Like [1] we solve the inverse problem by an iterative update using a finite difference approximation to the Jacobian. Starting with an initial guess of $\underline{p}$, the forward solution is used to calculate an endpoint $\underline{x}'$ for the proposed trajectory. The Jacobian J for the point $\underline{x}'$ is calculated by finite difference approximation using $h = 0.01$:

$$J_{i,j} = \frac{\partial \underline{x}_i}{\partial \underline{p}_j} = \frac{\underline{x}'_i - f_i(\underline{p}_j + h)}{h} \quad (13)$$

The difference $\Delta\underline{x}$ between $\underline{x}_T$ and $\underline{x}'$ and the inverse Jacobian can then be used to estimate $\Delta\underline{p}$ (Eq. 9). For stability in convergence, $\Delta\underline{p}$ is multiplied with a gain constant ($\mu = 0.2$), and added to $\underline{p}$ to produce a new trajectory:

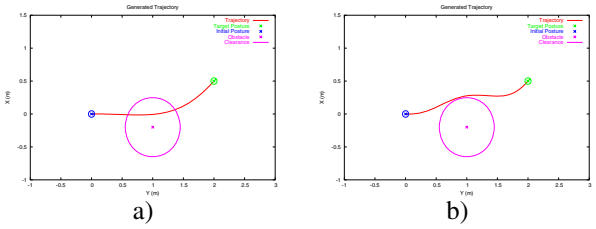
$$\underline{p} = \underline{p} + \mu\Delta\underline{p} \quad (14)$$

This process can then be performed iteratively and the trajectory $\underline{p}$ will converge to an acceptable trajectory.

4. Trajectory Generation Examples

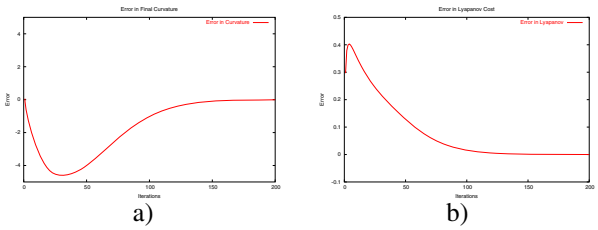
Fourth order curvature polynomial were used to generate trajectories for environments containing one or two obstacles. A trajectory was said to be acceptable when the following termination conditions were met:

$$\sqrt{(x_T - x(s_f))^2 + (y_T - y(s_f))^2} <= 0.01$$



a) b)

Figure 1. An example trajectory: a) initial cubic, b) fourth order trajectory.



a) b)

Figure 2. Convergence to end posture: error in a) curvature κ, and b) the cost function L over 200 iterations.

$$|\theta_T - \theta(s_f)| <= 0.1, |\kappa_T - \kappa(s_f)| <= 0.01$$
$$|L_T - L(s_f)| <= 0.005$$

4.1. Trajectories with One Obstacle

An example of a computed trajectory is shown in Figure 1. Graph a) shows the initial trajectory generated using a cubic curvature polynomial. The generated trajectory intersects with the clearance circle indicating the generated trajectory will result in collision. Graph b) shows the trajectory after it is adapted using the fourth order curvature polynomial. The trajectory now skirts the obstacle at the clearance distance resulting in a path with no collisions.

Figure 2 shows the convergence of the endpoint of the trajectory toward the target state at each step in the iterative estimation. All components of the endpoint converge to the desired values. To demonstrate continuity in the generated trajectory, the curvature over the path is shown in Figure 3.

In general for configurations where there was only one obstacle, if an initial cubic trajectory existed then a fourth order trajectory could be found that avoided the obstacle.

4.2. Trajectories with Multiple Obstacles

The proposed approach to trajectory generation can also deal with more than one obstacle in certain situations. Fig-

0-7695-2504-0/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

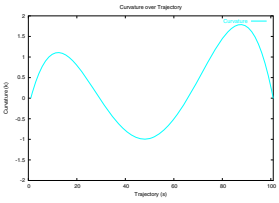
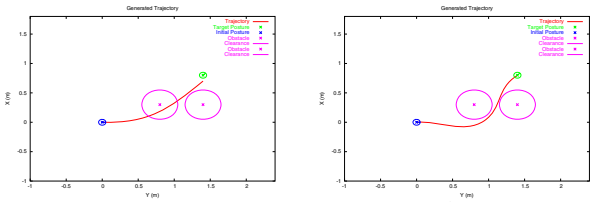


Figure 3. Curvature in fourth order polynomial trajectory over length of path.



a) b)

Figure 4. Trajectories with multiple obstacles: a) initial and b) generated trajectory for a narrow passageway

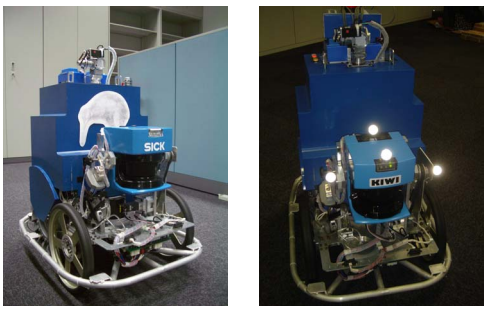
ure 4 shows an example path through an environment inhabited by two obstacles. The trajectories a) initial, and b)planned, show a path through two obstacles which are close together, such as might occur when a robot is navigating through a doorway.

4.3. Computation Time

The computation time for the example trajectories given above are given in Table 1. The trajectories were computed on a 3.06GHz processor. The increase in computation time between the cubic and fourth order curvature polynomials can be attributed to the extra calculation required for $L(s)$, integrating a function of both $x(s)$ and $y(s)$, as well as the numeric integration technique used to evaluate $L(s)$. The total computation however, remains quite low, and is sufficiently fast for real time trajectory generation and feedback based re-planning and control.

Table 1. Computation Time for Example Trajectories

Traj.	Cubic (ms)	4th Order (ms)	Total (ms)
Fig 1 b)	0.52	22.56	23.08
Fig 4 b)	0.14	33.79	33.93
Fig 4 d)	0.51	33.01	33.52



a) b)

Figure 5. The Toyota mobile robot with motion capture markers.

4.4. Limitations

Although the above algorithm converges to acceptable trajectories in the examples given above, the success of trajectory generation is subject to some considerable limitations. The main limitation is, of course that the method is a form of gradient descent and as such is subject to local minima, especially in the case of multiple obstacles.

Another related limitation is when obstacles are arranged in such a configuration so as the initial trajectory lies within the basin of an acceptable trajectory but that the possible trajectory requires more zero-crossings of curvature than are able to be modeled by the polynomial.

5. Experiments

The trajectory generation system was implemented on a mobile robot developed by Toyota Motor Corporation (shown in Figure 5 a)). The robot has four wheels, the front pair forming a differential drive, the rear passive and constructed of rollers to allow sideways motion. Thus the robot is not strictly car-like and can rotate on the spot. Experiments to validate the smooth trajectory curve generation were conducted within a $5 \times 5m$ area covered by a motion capture system. Motion capture markers were placed on the robot to recover the ground truth of robot motion (Figure 5 b)). In the experiments a goal is set manually, the robot senses objects with a laser range finder and plans a trajectory through the environment. While executing the trajectory localisation is performed within a predefined map.

5.1. Implementation

The localisation and planning system of the mobile robot is run off-board. Wireless networks are used to send sensor data off-board and to receive control instructions from the

0-7695-2504-0/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

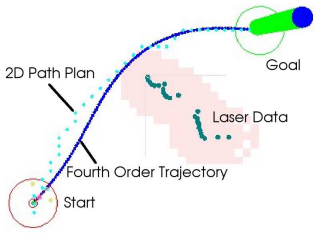


Figure 6. Setup for trajectory execution experiment.

planner. Sensor data (laser range, odometry) is used to localise the robot within a known map under a particle filter framework. Given the localisation result and a manually assigned goal localisation path planning is then performed. Path planning has three phases:

1. A* Grid Search: A two dimensional grid based planner is used to find the shortest path to the goal location. As long as a possible path exists, it will be found.

2. Cubic Trajectory Generation: Given a local sub-goal plan a cubic polynomial trajectory. Cubic generation can fail because of impossible configurations, singular matrices or divergence.

3. Fourth Order Trajectory Generation: If a cubic trajectory exists but has collisions, generate a fourth order polynomial trajectory. This can also fail due to impossible configurations, singular matrices or divergence.

Depending on which level of planning succeeds, one of three different types of control are sent to the robot: 1) a velocity control (v, ω), 2) a cubic control: the cubic coefficients (a, b, c, s), or 3) a fourth order control (a, b, c, d, s).

On receiving a control the robot converts the control into a velocity profile, smoothly transitioning from the current velocity and ensuring a smooth deceleration to zero after a given time for safety purposes. In the case of cubic and fourth order polynomial controls, the curve over distance s is converted into a velocity profile over time, with the constraints that the linear and steering velocities do not exceed their allowed maximum, and that curvature $\kappa(s)$ is preserved.

5.2. Trajectory Execution

To confirm the robot's ability to execute a continuous trajectory accurately the robot was commanded to execute a fourth order trajectory at a number of different velocities.

Figure 6 shows the experiment setup, the robot's start(red) and end (green) postures as circles, laser data points (dark green), the 2D planned path (light blue dots), and the planned fourth order trajectory (dark blue).

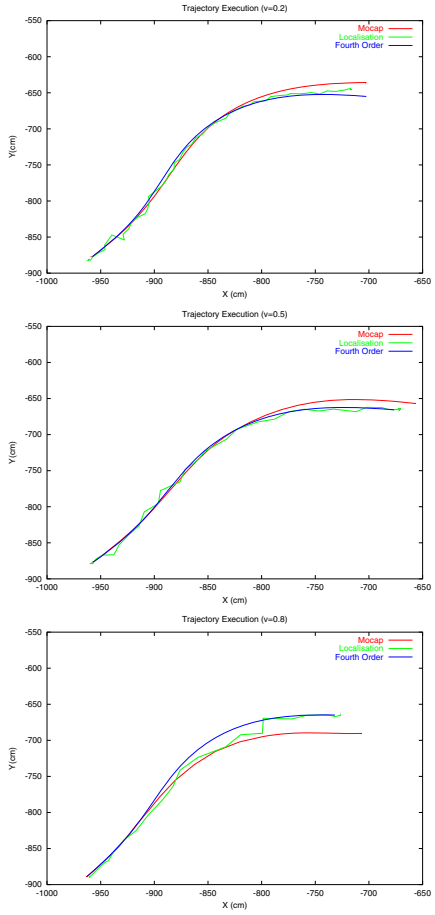


Figure 7. Curvature in fourth order polynomial trajectory over length of path.

Figure 7 shows the executed trajectories for robot maximum linear velocities of 0.2, 0.5, and 0.8m/s. The Robot follows the planned trajectories nicely, apart from drifting somewhat when the maximum velocity is 0.8m/s. Unfortunately at the end of the path the robot was approaching the boundary of the motion capture region, and the measure of the ground truth could also be drifting due to occlusions off markers. The trajectory execution results are good, given that no attempt to account for slippage of the passive back wheels was made.

5.3. Replanning

Given that drift does occur replanning while executing a trajectory is desirable. Localisation results can be used to determine how far the robot drifts from the planned trajectory and can trigger replanning when a threshold distance

0-7695-2504-0/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

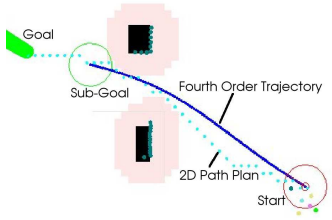


Figure 8. Initial planned fourth order trajectory for replanning experiment.

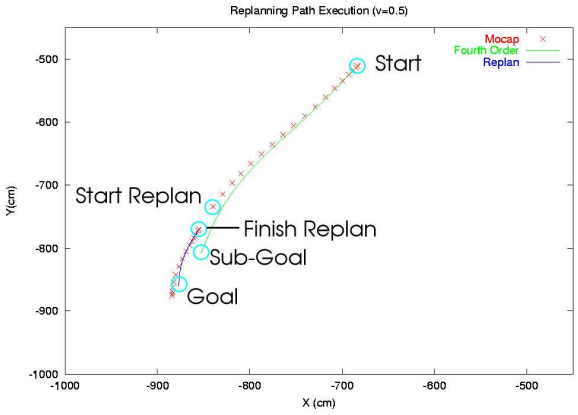


Figure 9. Replanning a trajectory.

is crossed (in these experiments 0.2m). Replanning is also triggered when approaching the end of a trajectory aimed at a sub-goal.

Figure 8 shows the setup for the replanning experiment. Figure 9 shows the motion capture data (red crosses) displaying the executed path. The initial planned trajectory is shown (green) as well a replanned trajectory (blue). There is a pause in motion capture data at the moment of replanning, due to the extra computation time, however the path remains continuous even after the replan.

The off-board loop, including receiving sensor data, localisation, planning and communications, takes about $200ms$. If replanning is not necessary this can drop to about $120ms$.

5.4. Extended Path

Figures 10 and 11 show the setup and the results of an experiment in which the robot executed an extended path, containing a number of separate planned trajectories. Figure 10 shows the obstacles in the environment: two boxes, which are included in the map and one cylindrical obstacle which isn't in the map. Figure 11 shows the motion capture data plotting the true path and the localisation data. The motion capture data is absent in some cases when the robot

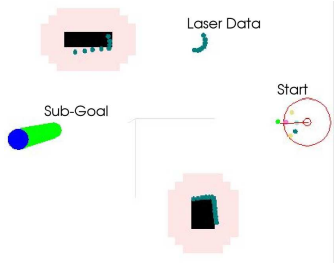


Figure 10. Setup of the experiment for extended path navigation.

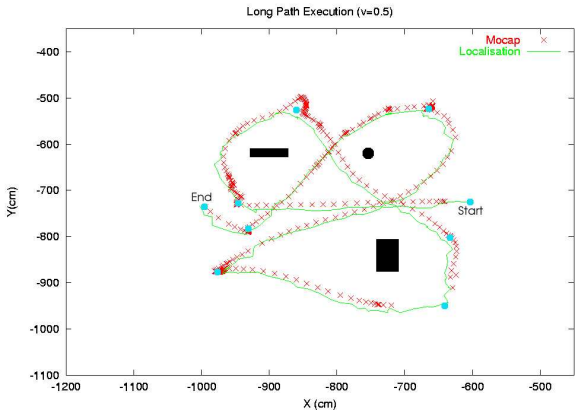


Figure 11. An extended robot path including many trajectories.

moves close to the edge of the motion capture region, most notably at the bottom right of figure. Although their are parts of the path which are not continuous in curvature (for example when the robot turns on the spot), there are some nice smooth curves illustrating the ability of the system to generate and execute continuous trajectories.

6. Conclusions

This paper has presented an extension to continuous curvature polynomial trajectory generation which explicitly considers the presence of obstacles. The extension entails adding a fourth order term to the previous cubic polynomial and a cost term to the state equations. Solving for the parameters of the polynomial given a start and end state then produces polynomial trajectories which minimise the cost term. Trajectories can also be generated quickly, typically in less than $40ms$, making this approach suitable for real time control. Trajectories were planned through narrow passages between obstacles without the need for subdivision or re-planning. Experiments in real time trajectory

0-7695-2504-0/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

generation and execution were conducted, verifying this approaches suitability for mobile robot control.

In future work, a method to improve the generation of initial trajectories is needed such as using a lookup table of predefined trajectories. Other work could include the use of a faster numeric integration method, extending the system to planning with dynamic obstacles, and the use of other cost functions such as minimal curvature.

References

[1] B. Nagy, A.Kelly (2001) Trajectory Generation for Car-like Robots using Cubic Curvature Polynomials, International Conference on Field and Service Robotics (FSR).

[2] J. Kuffner (2004) Effi cient Optimal Search of Euclidean-Cost grids and Lattices, International Conference on Intelligent Robotic Systems (IROS).

[3] C. Stachniss, W. Burgard (2002) An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments, International Conference on Intelligent Robotic Systems (IROS).

[4] J.-C. Latombe (1991) Robot Motion Planning, Kluwer Academic Publishers.

[5] H. Dellingette, M. Herbert, K. Ikeuchi (1991) Trajectory Generation with Curvature Constraint based on Energy Minimization, International Conference on Intelligent Robotic Systems (IROS).

[6] K. Konolige (2000) A Gradient Method for Realtime Robot Control, International Conference on Intelligent Robotic Systems (IROS).

[7] R. Arkin (1987) Motor Schema Based Mobile Robot Navigation, International Conference on Robotics and Automation (ICRA).

[8] D. Fox, W. Burgard, S. Thrun (1998) The Dynamic Window Approach to Collision Avoidance, International Conference on Robotics and Automation (ICRA).

[9] M. Khatib, H. Jaouni, R. Chatila, J. P. Laumond (1997) Dynamic Path Modifi cation for Car-like Nonholonomic Mobile Robots, International Conference on Robotics and Automation (ICRA).

[10] J. M. Wandosell, B. Graf (2002) Non-Holonomic Navigation System of a Walking Aid Robot, International Workshop on Robot and Human Interactive Communication (ROMAN) .

[11] O. Lefebvre, F. Lamiraux, C. Pradalier, T. Fraichard (2004) Obstacles Avoidance for Car-Like Robots, International Conference on Robotics and Automation (ICRA).

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY