Autonomous Vehicle Navigation

Scott A. Dial and Jason T. Gustafson

May 10, 2007

Contents

1 Objective ............................................. 2

2 Vehicle Control and Feedback ............................................. 2
   2.1 Physical State ............................................. 2
   2.2 Deriving Physical State ....................................... 2
   2.3 Dynamic Constraints ........................................... 3
   2.4 Point-to-Point Navigation ...................................... 3
   2.5 Speed Control ................................................. 4

3 Path Planning with Simple Splines ........................................ 4
   3.1 Method ....................................................... 4
   3.2 Drawbacks .................................................... 6

4 Conclusions .................................................. 7
1 Objective

Our objective is to create an autonomous driver and navigator of GPS waypoints. A desired course will be provided as a series of waypoints with speed restrictions. Our driver is to travel through these waypoints as fast possible.

2 Vehicle Control and Feedback

A large component of developing an autonomous navigator is low-level control of the vehicle and interpretation of feedback. Through the control interfaces provided by the cart, we are able to influence speed and wheel position, and we are responded back with actual speed and wheel position, along with current information from a GPS sensor. The most naïve implementation of control would ignore these sensor readings, and the most robust implementation would use them all. What follows is an attempt to use them all.

2.1 Physical State

The current state of the vehicle $V$ can ideally be represented by

$$V_s = \langle P, dP, S \rangle$$

where $P$ is the vehicles absolute position, $dP$ is the current change in position, and $S$ is the current sensor information.

With the information of $V_s$, we have all the information required to control our vehicle. At the lowest level, we can represent control as desired speed and wheel angle:

$$V_c = \langle s, w \rangle$$

2.2 Deriving Physical State

For deriving the state of the vehicle, the sensor array can be represented by

$$S = \langle s, w, P_{GPS}, dP_{GPS} \rangle$$

where $s$ is the actual speed, $w$ is the actual wheel angle, $P_{GPS}$ is the position indicated by GPS, and $dP_{GPS}$ is the change in position indicated by GPS.

The naïve implementation would assume $\langle P, dP \rangle$ equals $\langle P_{GPS}, dP_{GPS} \rangle$, however such a naïve assumption limits us greatly. The GPS information has a limited resolution and is updated at a reduced rate; if we solely used GPS information, we would also be out of luck if the satellites were unavailable. The best strategy to take here is to use the GPS information to aide the accuracy of a completely derived $V_s$.

If we assume we know the initial state of the vehicle, we can use the next $\langle s, w \rangle$ to derive the next state. Given $\langle s, w \rangle$, we assume we drove a circular path and calculate our new $P$. Additionally, we can calculate our new heading to obtain $dP$. We shall call this the predicted state.
If GPS information is available, then we can enhance the accuracy of our prediction through the use of a Kalman filter. For simplicity, we perform our filtering in a completely orthogonally state space. Each orthogonal component can be filtered separately as if they were one-dimensional filters.

A one-dimensional Kalman filter can be written as

\[ P_x = P_x + Q_x \]  \hspace{1cm} (4)
\[ K_x = \frac{P_x}{P_x + R_x} \]  \hspace{1cm} (5)
\[ x = x_p + K_x(x_m - x_p) \]  \hspace{1cm} (6)
\[ P_x = (1 - K_x)P_x \]  \hspace{1cm} (7)

where \( x \) is the variable we are filtering, \( P_x \) is the process noise, \( Q_x \) is the measurement variance, \( R_x \) is the process variance, \( K_x \) is the Kalman gain, \( x_m \) is the measured value, and \( x_p \) is the predicted value. We must then provide the variances for both the derived information (predicted) and GPS information (measured).

### 2.3 Dynamic Constraints

As our vehicle begins to travel at faster speeds, the impact of aggressive turns will begin to exceed the limits of safety. Although there has not been sufficient testing to find a critical limit, we could theoretically execute a 1 G turn which is believed to be in excess of safety. At this point in time, the apparent limit of comfort is at a little over 0.6 G. To ensure we do not exceed this limit, we must dynamically constrain \( V_c \) for a given \( V_s \).

We will limit the wheel angle at any given moment by

\[ \omega_{max} = \frac{A_{max} L_{ub}}{s^2} \]  \hspace{1cm} (8)

where \( A_{max} \) is the maximum lateral acceleration allowed, \( L_{ub} \) is the distance between the front and rear wheels, and \( s \) is our current speed.

Restricting steering based on speed has the unfortunate side-effect that if we need to turn, we won’t be allowed. To counteract this problem, we also limit speed based on desired wheel angle by

\[ s_{max} = s_m \omega + s_b \]  \hspace{1cm} (9)

where \( s_m \) and \( s_b \) are inputs to a linear model of speed in relationship to \( \omega \), wheel angle.

### 2.4 Point-to-Point Navigation

Navigation of a vehicle can be reduced to traversing from one absolute position to another. The process by which one gets from one point to another could be executed a number of ways. The simplest method, and the one used by our navigator, is to find the heading of the line between the two points and seek to set out heading to match. This method will constrain us to the straight-line path between the two points with the caveat that our corridor constraint is conical.
2.5 Speed Control

In the process of navigating, we will need to control our speed. In a simple point-to-point navigator, we can simply look forward to the next waypoint to find a desired speed and calculate backwards in time our maximum speed to allow that desire to be realized. We apply rudimentary physics to find this maximum:

\[ v_{\text{desired}}^2 = v_{\text{max}}^2 + 2a_d d \tag{10} \]

, where \( a_d \) is our maximal amount of deceleration possible and \( d \) is our distance from the point of interest.

In the general case, the distance between two points may not provide for enough deceleration if we are coasting. To achieve our desired speeds, we need to look several points ahead to find the point that actually provides the constraint. For a known course, we can walk through the points backwards to find these speed constraints easily.

3 Path Planning with Simple Splines

The problem of following a collection of waypoints was the primary task we faced in the course. The simplest solution is to use a straight path algorithm to hit each point in turn. As long as the waypoints are chosen well, this solution actually does a good job. It is worth consideration though what advantages are gained by constructing a smooth path for following the points. For one, the cart would potentially be able to hit each waypoint at a higher velocity than otherwise. The cart might also require some anticipation of the waypoint beyond the next immediate point to ensure that it turns within the corridor. Smoothing the path could accomplish this. In any case, there is definitely cause to consider methods for achieving higher orders of continuity in the path the cart is to follow. One such method is an interpolating spline. Below is a summary of our attempt to implement this and the problems that we had to deal with.

3.1 Method

The first spline we attempted is known as the Kochanek-Bartels spline after its creators. It uses the Hermite basis functions which are derived easily from the following equations, in which \( \{P_i\}_{i=1}^n \) are the points to be interpolated and \( \{D_i\}_{i=1}^n \) are the derivatives to be specified at the waypoints.

\[
F_i(t) = at^3 + bt^2 + ct + d \\
F_i(0) = P_i, \ F_i(1) = P_{i+1} \\
F'_i(0) = D_i, \ F'_i(1) = D_{i+1}
\]

The spline is constructed piecewise between each of the points. To compute the equation of the curve between \( P_i \) and \( P_{i+1} \), we just have to multiply the Hermite basis matrix \( \mathbf{H} \) by the vector \( \mathbf{C} = (P_i \ P_{i+1} \ D_i \ D_{i+1})^t \).

4
\[ F_i(t) = \mathbf{T} \cdot \mathbf{H} \cdot \mathbf{C} \]

\[ = (t^3 \quad t^2 \quad t \quad 1) \cdot \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} P_i \\ P_{i+1} \\ D_i \\ D_{i+1} \end{pmatrix} \]  

(11)

(12)

The only question is how to determine the points \( \{D_i\}_{i=1}^n \). With this particular spline, three parameters are specified to control the shape of the curve: tension \( t \), continuity \( c \), and bias \( b \). The following equations are used for the vectors \( D_i \) and \( D_{i+1} \).

\[
D_i = \frac{(1 - t)(1 + c)(1 + b)}{2} \cdot (P_i - P_{i-1}) + \frac{(1 - t)(1 - c)(1 - b)}{2} \cdot (P_{i+1} - P_i)
\]

\[
D_{i+1} = \frac{(1 - t)(1 - c)(1 - b)}{2} \cdot (P_{i+1} - P_i) + \frac{(1 - t)(1 + c)(1 - b)}{2} \cdot (P_{i+2} - P_{i+1})
\]

The ability to directly control the shape of the curve was our primary motivation for using this spline, as we were hoping to limit the curvature so that we wouldn’t plan a course the cart could not follow. We tested various values for the three parameters, but they did not give us the control we hoped for and we ended up settling on values of 0 for each of them. This essentially reduces the spline to the Catmull-Rom spline with \( D_i \) and \( D_{i+1} \) defined below.

\[
D_i = \frac{P_{i+1} - P_{i-1}}{2}
\]

\[
D_{i+1} = \frac{P_{i+2} - P_i}{2}
\]

This of course means that the extra points \( P_0 \) and \( P_{n+1} \) are also required to complete the interpolation. This is actually useful because it gives us a simple way to specify an exact value for the derivative at the first point (the initial velocity of the cart). If we require \( V_0 \) to be the derivative at the point \( P_1 \), we choose \( P_0 \) in the following way to ensure it.

\[
P_0 = P_2 - 2 \cdot V_0 \]

(13)

This is pretty much all that is needed to state the algorithm which we used to plan the path. One difficulty that we had to face was how to follow the path once it was computed. To our disappointment, the only solution we could come up with was to add extra points along the curve and then follow them one at a time. This presented difficulties as will be explained later. For now, we state the algorithm, which is actually quite simple. It is assumed that the cart is at some initial location \( X_0 \) moving at velocity of \( V_0 \) and that the waypoints are \( \{P_i\}_{i=2}^n \).

1. First we set \( P_1 \) to the initial position \( X_0 \). The equation above is used to set \( P_0 \). \( P_{n+1} \) can be set to \( P_n \). Also set \( i = 1 \).
2. Construct the spline points between \( P_i \) and \( P_{i+1} \) by first computing \( M = H \ast C \) as above. To get \( k - 1 \) spline points, we now just have to compute \( F_i(t) = \left( t^3 \ t^2 \ t \right) \cdot M \) for \( t = 0, \frac{1}{k}, \frac{2}{k}, \ldots, 1 \).

3. If \( i = n \), then we’re done. Otherwise, increment \( i \) and repeat the second step.

The algorithm efficiently computes the points on the curve for the cart to follow. These points are guaranteed to include the waypoints themselves. Thus we can use these points to follow a smooth path through all of the points. And since it explicitly accounts for the current location and velocity of the cart, it can be recomputed at any time with whatever waypoints remain. But there are some serious drawbacks to the method which are detailed below.

### 3.2 Drawbacks

One of the primary drawbacks of the splining technique we used is that it does not give us enough control over the shape of the spline. The curvature constraints we hoped could be handled through the parameters of the tcb-spline are definitely not satisfied. So it’s perfectly possible that the method will compute a path which the cart will find impossible to follow. Our solution to this problem was simply to watch for spline points that the cart could not hit and to recompute the spline at that point. This is actually not too serious a drawback since we assume that the waypoints are specified so that the cart will actually be able to hit them without any complicated maneuvers.

Another problem with the shape of the curve is that there is no way that we can guarantee that it remains within the corridors between the waypoints. This wasn’t a primary consideration until late in the semester, so we didn’t come up with a solution. One of the drawbacks of the Hermite curves in general is that they are not necessarily contained within the convex hull of the points they interpolate, so there is some potential for the curve to fall well outside a corridor. A B-spline, on the other hand, does have the property that the curve it generates is contained within the convex hull of the points given. There are some details to work out, but it seems possible that if extra points are chosen carefully, then the convex hull of a B-spline might be constructed so that it itself is contained within corridors of the points. So the problem of staying within the corridor is reduced to selecting extra control points which force the convex hull into the corridor. This may actually be a difficult problem, but it seems to be worth considering if a spline solution is desired.

The more serious problem with our splining solution is the method we used to make the cart follow the curve. As was mentioned above, we simply added more points for the cart to hit in turn. Between each of these points, we were still using the straight path algorithm. The problem is that when a spline point is reached, the cart is not necessarily heading in the right direction for the next point. In most cases, it is close, but with every point hit, there is still a noticeable jerk in the cart’s movement as it turns to hit the next spline point. Our solution to this problem was to increase the number of spline points between the waypoints so that we would follow the curves more closely. We only ran a couple tests with this idea, but inserting spline points every meter seemed to eliminate the most noticeable effects of the spline approximation. This forced us to consider the
distance between the waypoints when computing the spline, but the algorithm above can be easily modified to do this. One of the fundamental problems that we did not solve, but which would certainly aid in coming up with a better solution to the problem of following a curve, is how to make the cart arc into a waypoint. With our solution, whenever we hit a waypoint, the wheels are heading more or less straight forward. This obviously constrains the continuity of the slope of any path we try to follow. If the cart was smart enough to hit a point with the wheels turned toward the next point, the jerks in the movement would be reduced. But, one of the reasons we didn’t make much progress on this problem, is that it seems very difficult.

4 Conclusions

Using a spline to plan a path for the cart is definitely doable and its performance is roughly on par with the simple solution of using the straight path algorithm to simply hit the waypoints one after the other. It remains to be seen whether the extra effort put into computing and following the path is justified though. Certainly the method will have to be modified if corridor constraints are to be satisfied and a more sophisticated method for following the curve would be desirable. If these can be dealt with, it is possible that the advantages to having a smoothed path will be more apparent than it was in our testing.