Reactive Trajectory Planning and Tracking for Pedestrian-Aware Autonomous Driving in Urban Environments

Robert G. Cofield[1] and Rakesh Gupta[2]

Abstract— **In this paper, we address the problem of trajectory planning while simultaneously reacting to the presence of pedestrians for an autonomous car on urban roads. Past work limits jerk, velocities and acceleration for smooth trajectories, without considering reactive behaviors such as responding to pedestrians. Other systems based on collision avoidance, plan paths around obstacles and pedestrians in unstructured environments. In this paper, we present an integrated trajectory generation and tracking system. Our system simultaneously considers both parameter and reactive constraints for smooth trajectory and updates it in real-time.**

We present a novel online method for planning trajectories to follow a given urban path while honoring traffic regulations such as stop signs at intersections. We update the trajectory to safely avoid pedestrians on the road by slowing down or stopping. Our method has closed form solutions, runs at 20 Hz, and is efficient and reliable for use in online planning. We have confirmed this with a test vehicle and pedestrians with over 100 hours of testing under driverless operation.

I. Introduction

An autonomous ground vehicle operating on roadways typically plans trajectories with the goal of minimizing navigation time. It is common to employ a path-velocity decomposition to separate the twin tasks of Path and Trajectory Planning, so that these tasks can be run sequentially or in parallel. *Path Planning* computes the optimal path while *Trajectory Planning* computes a time-parameterized velocity profile appropriate for the scenario.

Trajectory computation for traveling on the path is often guided by passenger comfort. For human passengers, it is widely believed that high levels of jerk (time derivative of acceleration) and high acceleration and velocity values are prime contributors to ride discomfort. The intended speed, acceleration, and time derivatives of position are also subject to constraints due to legal speed limits, engine dynamics, desired side-slip limits, traction availability, and rollover risk. Lot of work has been performed on optimizing these constraints for highways and country roads [1]–[4]. Similar work performed by Bianco et al [5]–[9] sets the travel time a priori and optimally minimizes jerk without hard limitations or direct jerk control. Other work on optimizing these constraints on urban roads [10], [11] doesn't consider reactive behaviors such as responding to pedestrians and other dynamic objects in the scene.

* This work was performed at Honda Research Institute USA.

[1] Auburn University GAVLab, 1418 Wiggins Hall, Auburn, AL 36832. *robertgcofield@gmail.com*

[2] Honda Research Institute USA Inc, 375 Ravendale Drive, Mountain View, CA 94043. *rgupta@hra.com*

Fig. 1. Showing car used in our experiments.

The set of possible paths to the destination is typically restricted by constraints such as available travel lanes, and obstacles including pedestrians. Many systems in the literature plan paths around pedestrians, stop for them, or produce a warning [12]–[16]. These systems make an assumption that alternative paths around obstacles are available. Such an assumption may not always be valid in an urban scenario. Planners that optimize human comfort by modifying the path to go around obstacles [17], [18] do not feature stopping as a well-integrated behavior. In this paper, we describe a trajectory generation system that satisfies the constraints for human comfort while being reactive to the presence of pedestrians on the road. We enforce hard upper limits on jerk without requiring that travel time be known for planning.

Our autonomous car should travel autonomously from start to destination while honoring traffic laws and avoiding pedestrians on the roadway. At stop signs, it should stop and wait for an appropriate amount of time. The car should also stop, slow down for pedestrians at the intersections as well as jay walking pedestrians. Further, the pedestrians can be static or moving. Pedestrians on the sidewalks should be ignored by the car.

In this paper, we present a deterministic autonomy framework for pedestrian-aware ground vehicles with a novel method for planning longitudinal trajectories. Our method segments the path using both kinematic and dynamic constraints. Each segment is then planned sequentially by choosing a piecewise constant jerk profile. We intelligently choose a jerk profile from a set of analytically solved profiles such that acceleration is continuous and within limits throughout the entire path. The resultant trajectory plan is sampled to provide a reference for real-time vehicle control.

Section II describes the architecture of our system and interacting subsystems. Section III describes the high-level Path Manager that calls the trajectory planner to exhibit reactive behavior and stop for pedestrians on the road. Section

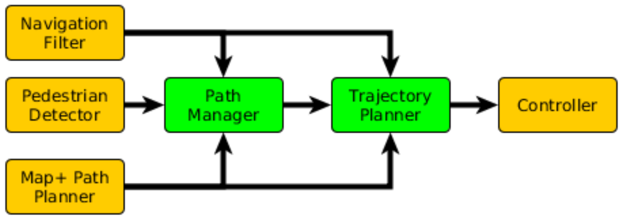


Fig. 2. Primary system components. The *Path Manager* and *Trajectory Planner* components (in green) are presented herein.

IV describes trajectory planning to generate a trajectory that satisfies maximum velocity, acceleration and jerk constraints. Once the trajectory is formulated, we send the reference signal to the controllers that govern steering, braking, and engine speed. This is trajectory tracking and is discussed in Section V, followed by results and conclusions.

II. System Architecture

Our system architecture is shown in Figure 2. We have three major components: a High-Level Planner, a Low-Level Planner, and a Software Controller. The High-Level Planner is referred to as the *Path Manager*, and performs decision-making as well as governs operation of the low-level planner. Its primary functions are:

- Stop the vehicle at stop signs and wait until the associated intersection is clear before continuing.
- Reactively reduce speed or stop altogether for pedestrians which pose a collision risk, then continue when they are clear.
- Govern the Trajectory Planner.

The Low-Level Planner is also referred to as the *Trajectory Planner*, and is responsible for quantifying the actions dictated by the Path Manager. Its primary functions are:

- Compute medium-range trajectories for a given path.
- Revise trajectories to react to pedestrians.
- Sample the current trajectory for input to the controller.

The Software Controller governs engine speed, braking, and steering. It takes as an input a trajectory sampled at 10 Hz spanning 2 seconds, beginning at the moment of data transmission. This short-term trajectory is oriented in the vehicle body frame. This paper focuses on the Path Manager (Sec. III) and the Trajectory Planner (Sec. IV and Sec. V).

Beyond these major components, our system has four primary input components: Navigation Filtering, Pedestrian Detection, a Path Planner, and a Map. For Navigation Filtering, we use an ADMA-G commercial automotive GPS/INS system. It receives RTK corrections via cell modem. It outputs global position, global heading, body frame velocity and body frame acceleration to centimeter level accuracy.

Pedestrian Detection is performed via a camera and LiDAR system based on [19], [20]. The pedestrian detector outputs a bounding box for each pedestrian as shown in Figure 3. The pedestrian position is computed in the vehicle body frame, and subsequently transformed to the global

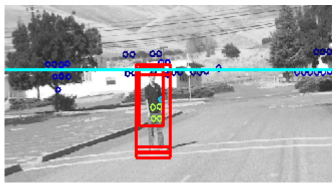


Fig. 3. Showing detected pedestrian by the computer vision system

frame to be related to the planned path. It is assumed that no unique identifier is available to differentiate one pedestrian from the other in a single epoch, or to correlate the same pedestrian between epochs.

We assume access to a lane level Path Planner. For example, this can be a service similar to Google Maps except that it provides a lane level route instead of a road level route. We assume that the path planner operates independently from the trajectory planner and tracker. Using the path planner, we obtain a path along lane centers from the vehicle's current location (computed by the navigation filter) to the user specified destination. This plan is output as a set of global waypoints.

We also assume a High-Resolution Digital Map with lane centers, stop signs, crosswalk geometry, and turning curves at intersections. This map provides the location of stop signs. We also have an In-Out Algorithm implemented via polygon intersection that can be used to ignore safe pedestrians Pedestrians that are on sidewalks, behind planned stops, and off-path crosswalks.

III. Path Manager

The Path Manager performs trajectory planning over an incremental window within the full path. This is done by breaking the planned path into smaller sub-paths at each stop location which are known a priori, as shown in Fig. 5. Trajectory planning is then performed for each sub-path incrementally as the vehicle progresses.

Prescribed initial and end conditions for the position, speed, and acceleration must be honored. All planning is longitudinal; that is, speed profile generation is the primary focus. We assume that the control subsystem ensures that the vehicle follows the prescribed path. This allows all planning to be done in a Frenet (rather than Cartesian) frame for simplicity. The Frenet axis which runs along the length of the planned path is denoted by s, and all planning is performed in the s-axis. Differences along this axis (Δs) correspond to distances the car will actually travel. Mapping back to Cartesian coordinates (necessary for trajectory tracking) is well-discussed in the literature.

Our Path Manager is implemented as a discrete state machine, depicted in Fig. 4. In the absence of pedestrians, the path manager cycles between NORMAL and PSTOP states. In NORMAL state, the path manager sends the first subpath to the trajectory planner, which then plans and executes it, coming to a stop at the end of the subpath. In PSTOP mode, the path manager waits for a pre-set time duration before returning to NORMAL mode, where the next subpath is

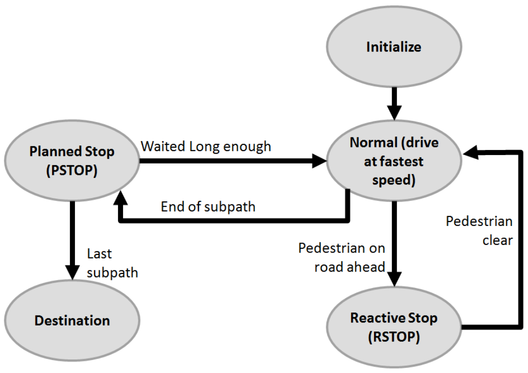


Fig. 4. State Transition Diagram for Path Manager. System transitions from NORMAL to PSTOP mode at each stop sign. In the presence of a pedestrian the system transitions to RSTOP.

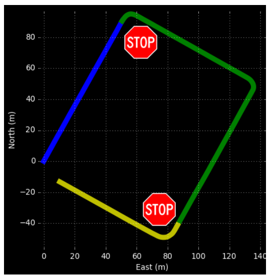


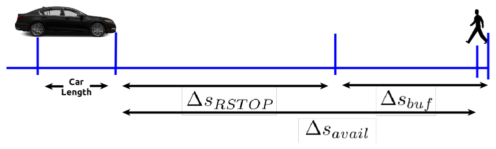
Fig. 5. Dividing the path into sub-paths at stop signs. The Trajectory Planner is serially called for each sub-path shown in different color.

executed. When pedestrians are present, the transition from PSTOP to NORMAL is delayed until no pedestrians are detected in the range denoted by Δs_{resume} as shown in Fig. 6(c).

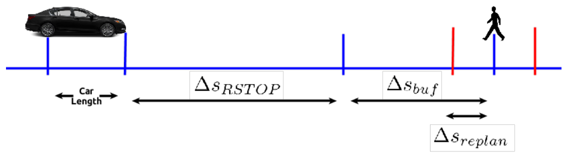
All detected pedestrian locations are compared against the high-resolution digital map. Off road pedestrians are not considered. Each pedestrian location is perpendicularly projected onto the path to determine the travel distance separating them from the car. Pedestrians that are on a different subpath than the vehicle, or behind the vehicle are disregarded (i.e., $s_{ped} < s_{veh}$). All the remaining pedestrians pose potential collision risk. However, only the closest pedestrian is considered for RSTOP.

The distinction between momentarily braking for a pedestrian that quickly leaves the roadway and coming to a full stop needs no separate logic. They follow the same state transition, and since all trajectory plans are smooth, riders are rarely aware of the change. There is a brief waiting period before resuming NORMAL mode from RSTOP to ensure no tracking errors in pedestrian detection.

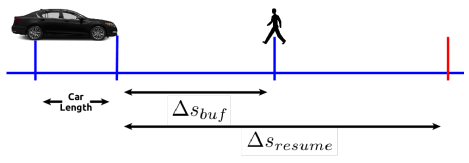
When a pedestrian is detected, the nominal distance needed to stop, Δs_{RSTOP}, must be computed given the vehicle's current location, speed, and forward acceleration. This serves as a dynamic buffer between the pedestrian and the vehicle. In order to calculate Δs_{RSTOP}, a hypothetical trajectory plan is created (but not implemented). A 3-phase profile using the vehicle's current state as initial conditions and $v_f = 0$ is created, and the resultant distance then



(a) In NORMAL mode, RSTOP is now triggered.



(b) RSTOP diagram showing bounds for revising the RSTOP trajectory.



(c) PSTOP diagram showing bounds for reverting to NORMAL mode.

Fig. 6. Graphical representation of static and dynamic thresholds used to trigger state transitions.

becomes the nominal required stopping distance.

Note that a static buffer distance, Δs_{buf} is added to ensure safety and comfort. RSTOP is triggered when $\Delta s_{avail} \leqslant \Delta s_{RSTOP} + \Delta s_{buf}$ and the actual available stopping distance $\Delta s_{avail} = s_{ped} - s_{veh}$ is computed. Planning and executing the trajectory for coming to a halt is detailed in Sec. IV-C.

This procedure is repeated when the nearest pedestrian location changes by a value more than some static buffer $\Delta s_{replan} < \Delta s_{buf}$ compared to the location for which the current RSTOP planned. Normal operation is resumed when the distance between the vehicle and the closest detected pedestrian satisfies $s_{ped} - s_{veh} > \Delta s_{resume} + \Delta s_{RSTOP}$. The static resume buffer should be set such that $\Delta s_{resume} > \Delta s_{buf} + \Delta s_{replan}$. These quantities are depicted graphically in Fig. 6. Nominal parameter values are: $\Delta s_{buf} = 8.5m$, $\Delta s_{replan} = 1.0m$, $\Delta s_{resume} = 12.5m$.

IV. Trajectory Planning

The task of Trajectory Planning is to take a set of scalar waypoints in the frenet frame from the Path Manager, and compute a series of constant jerk intervals for longitudinal motion. The resulting trajectory should minimize time, and satisfy all the constraints on speed, acceleration, and jerk. Integrating piecewise constant jerk over time yields acceleration which is piecewise linear over time, a speed which is piecewise quadratic over time, and position which is piecewise cubic over time.

A. Path Segmentation

In this section, we describe our approach to fitting appropriate jerk profiles to complex sub-paths. We first break each sub-path into segments where the constraint values can be considered constant. Then each segment can be compared to a library of possible jerk profiles to find a good match. Each segment is defined by the vector $\mathbf{b} = [v_i, a_i, L, v_m, v_f]$, with parameters initial speed, initial acceleration, total distance, speed limit, and the final speed, respectively. Additionally, the desired positive and negative acceleration $\mathbf{a}_m = [a_m^+, a_m^-]$ and the desired positive and negative jerk $\mathbf{j}_m = [j_m^+, j_m^-]$ are defined for each segment. For this work, they are set constant for all segments. However, this functionality enables behaviors such as more sedate driving in a school zone or when traversing small areas with lower friction coefficients, such as patches of ice or gravel detected via camera. A sub-path is then a set of N segments, each of which is defined by a set of parameters $\mathbf{b}_k$, $\mathbf{a}_{m,k}$, and $\mathbf{j}_{m,k}$ where $k = 1, ..., N$ and can be solved individually in sequence.

The segmentation process proceeds as follows:

1) *Ceiling Calculation*, where v_m, $\mathbf{a}_m$, and $\mathbf{j}_m$ are found for every point in the given sub-path.
2) *Clustering*, in which sub-path points are grouped sequentially and the segment boundaries are determined.
3) *Consolidation*, in which a single $\mathbf{b}$, $\mathbf{a}_m$, and $\mathbf{j}_m$ is selected for each segment.

The *Ceiling Calculation* is a minimax problem. Each of the 9 scalars in $\mathbf{b}$, $\mathbf{a}_m$, and $\mathbf{j}_m$ is a minimum of several maxima computed from an arbitrary set of constraints. For instance, acceleration constraints may include a set of limits derived from friction coefficients at every point along the sub-path.

For the purposes of developing a simple example application, two constraints are used for v_m: legal speed limit SL and lateral acceleration limit a_y^{max}. Lateral acceleration constraints are translated into longitudinal speed constraints using the approximation:

$$v_{m,w}(a_y) = \sqrt{|a_y^{max}/\kappa_w|}\ ,\ \ w = 1, ..., W \tag{1}$$

where W is the number of waypoints in the sub-path, and κ_w denotes the path curvature at each point. For each sub-path point, the value $v_{m,w}(a_y^{max})$ denotes the speed necessary to obtain the limit lateral acceleration a_y^{max}.

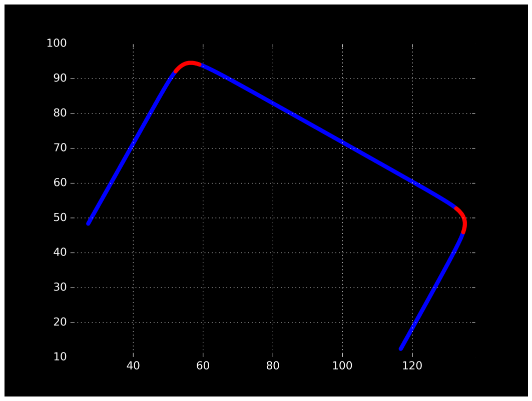
This information is used to choose the segment boundaries in the *Clustering* step. A prudent choice of segment boundaries is arguably a critical piece of the trajectory planning process. We choose a simple univariate heuristic which effectively identifies curves in the roadway using only the speed ceiling. Each time $v_{m,w} < SL$ becomes true, a curve is begun, and a segment boundary is drawn. Each time $v_{m,w} = SL$ is again true, the current curve is ended, and another segment boundary is drawn.

Figure 7 shows the result of this Clustering algorithm. We start with a sub-path similar in structure to the green sub-path in Fig. 5. This sub-path is broken up into five segments as shown in Fig. 7(a). The three segments shown in blue correspond to more straight portions, and the two shown in red correspond to more curved portions.

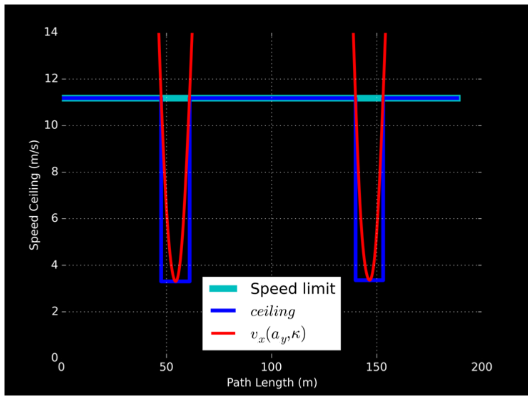
In the consolidation step, a single set of $\mathbf{b}$, $\mathbf{a}_m$, and $\mathbf{j}_m$ must be calculated for each segment using the values corresponding to the waypoints comprising the segment. The principle task is to find a value of v_m for each segment using the values from segmentation, SL and $v_{m,w}(a_y^{max})$. The value of the speed ceiling for segment k is chosen to be:

$$v_{m,k} = min\left(\mathbf{SL}_k, \mathbf{v}_m(a_y^{max})|_k\right) \tag{2}$$

where $\mathbf{SL}_k$ is the vector containing the values of the legal speed limit at every way-point in segment k. The vector $\mathbf{v}_m(a_y^{max})|_k$ contains the values of the speed limit from lateral acceleration at every way-point in segment k. While not time optimal, choosing the minimum of these values over the entire segment ensures that no constraints are violated. As a result, many of the curves have a constant speed. For the sake of simplicity, the values of $a_{m,k}^+$, $a_{m,k}^-$, $j_{m,k}^+$, and $j_{m,k}^-$ are initially set constant for the entire path, using values from [21]–[23]. Once $v_{m,k}$ are set for every segment in the



(a) The result of dividing a single sub-path into separate segments. The clustering algorithm used here sets segments boundaries before and after curves. The sub-path segments in red indicate areas where the longitudinal speed required to adhere to lateral acceleration limits is below the legal speed limit of 11.1 m/s (25 mi/hr).



(b) An example of how the sub-path's segment boundaries are determined, as well as how v_m is consolidated to determine the speed ceiling for each segment. The x-axis corresponds to distance along the subpath above. The legal speed limit of 11.1 m/s (25 mi/hr) is shown in cyan. The longitudinal speed required to adhere to lateral acceleration limits is shown in red. The resultant speed ceiling (v_m) is shown in blue.

Fig. 7. Dividing a subpath into segments and determining speed ceilings for each segment.

```
1:  procedure SegmentBoundarySpeeds($v_m$)
2:      $\mathbf{v}_p \leftarrow \mathbf{v}_m$                    ▷ Initialize targets as ceilings
3:      $\mathbf{v}_{p,0} \leftarrow v_i$                  ▷ Set starting speed to current
4:      $\mathbf{v}_p = [\mathbf{v}_p, 0]$                    ▷ Add final speed of 0
5:      for $i = 1, ..., N-1$ do
6:          if $v_{m,i+1} > v_{m,i}$ then
7:              $v_{p,i+1} \leftarrow v_{m,i+1}$
8:          else
9:              $v_{p,i+1} \leftarrow v_{m,i}$
10:         end if
11:     end for
12: end procedure
```

Fig. 8. Algorithm to set speed boundary conditions for interior segments. Having consolidated the waypoints into a set of path segments, the maximum allowable speed for each segment, $v_{m,i}$ is known. This algorithm uses the set of $\mathbf{v_m}$ for all segments to choose the highest allowable start and end speeds for each path segment.

subpath, boundary conditions are set to ensure that speed is continuous between segments. Acceleration continuity is addressed later. The algorithm in Fig. 8 is used to compute the vector $\mathbf{v}_p$ and each segment k is solved using:

$$v_{i,k} = v_{f,k-1} = v_{p,k} \tag{3}$$

The values of a_i and v_i for the first segment are set to the value currently reported from sensor data. Since the final acceleration is set to zero for all profiles, each subsequent segment begins with $a_i = 0$. For the final segment, setting $v_f = 0$ is prudent unless the scenario demands otherwise.

Now that all segments within the subpath have been parameterized, they are solved sequentially. A jerk vs. time profile is fit to each segment using the algorithm described in the next sub-section.

B. Fitting Jerk Profiles to each Segment

To introduce the process of computing a jerk profile for each segment, consider the case where a vehicle needs to travel an arbitrary distance ahead. Starting from an initial speed, the vehicle needs to accelerate to a traveling speed, maintain that speed for some distance, then brake to a stop at a prescribed end location. Figure 9 shows this trajectory over time.

Final forward acceleration is set to zero. A profile with M constant jerk phases (7 in this case) can be generated by solving for the time intervals $[\Delta t_1, ..., \Delta t_M]$. As we show later, $\mathbf{a}_m$, and $\mathbf{j}_m$ must sometimes be modified to obtain non-negative possible time intervals. To enable this, ranges of acceptable longitudinal jerk and acceleration $\mathbf{a}_m^{max} = [a_m^{+,max}, a_m^{-,max}]$ and $\mathbf{j}_m^{max} = [j_m^{+,max}, j_m^{-,max}]$ are set according to the relation in Eqs. 4 and 5. In the simplest case, one value for each of the quantities expressed in these relations is chosen for the entire path.

$$a_m^{+,max} \geqslant a_m^+ > 0 > a_m^- \geqslant a_m^{-,max} \tag{4}$$

$$j_m^{+,max} \geqslant j_m^+ > 0 > j_m^- \geqslant j_m^{-,max} \tag{5}$$

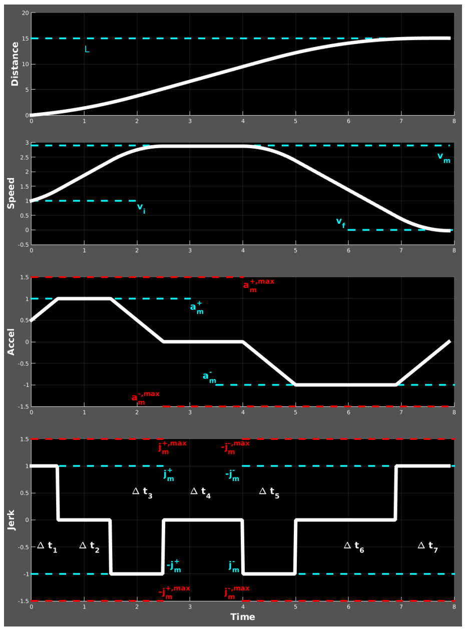
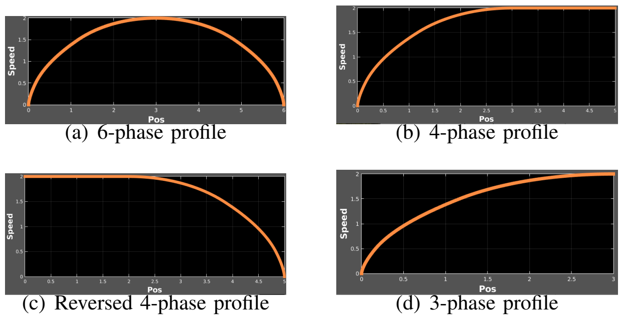


Fig. 9. 7-phase constant jerk profile along the curvilinear distance axis. All values needed to completely parameterize a single path segment for trajectory are labeled here.

The 7 phase solution is time optimal as it drives the car at the speed limit. However, sometimes the distance is too small to reach the speed limit, or the boundary conditions are different, such as going from a specific speed to a stop. We define four other basic profiles to cover all feasible values in $\mathbf{b}$, and our algorithm judiciously chooses the appropriate profile. For instance, if L is too short or v_m is too high, then it will not be possible to ever reach v_m with realistic values of $\mathbf{a}_m$ and $\mathbf{j}_m$. All impossible scenarios are automatically ruled out (e.g., end speed above speed limit, traveling speed of zero, negative speeds in $\mathbf{b}$, etc). Traveling backward (i.e., in reverse gear) is not supported in this work. To find closed-form solutions for $[\Delta t_1, ..., \Delta t_M]$, given $\mathbf{b}$, $\mathbf{a}_m$, and $\mathbf{j}_m$, a symbolic solver is used The five jerk profiles in our *Jerk Library* are:

1) 7-phase: This profile, with Δt_i as shown in Figure 9, is always attempted first. All time steps have a single solution, enabled by the fact that the initial and end conditions are known, and the two peak accelerations and two jerk magnitudes are provided as configuration parameters. In the case where Δt_2 or Δt_6 are negative, accel-jerk tuning must be employed. The 7-phase profile can be expressed as $(j_1 > 0, j_2 = 0, j_3 < 0, j_4 = 0, j_5 < 0, j_6 = 0, j_7 > 0)$. When Δt_4 is negative, the total segment length L is too short to reach top speed of v_m, so a 6-phase profile must be used.

2) 6-phase: In the case mentioned above, the speed limit is too high for a solution to exist for a 7-phase profile. The middle travel period Δt_4 is removed, so that the vehicle increases speed and then immediately decreases speed. The peak speed, in this case, has a solution that cannot be manipulated directly. Without this known value, the problem



(a) 6-phase profile (b) 4-phase profile (c) Reversed 4-phase profile (d) 3-phase profile

Fig. 10. 6, 4, Reversed 4 and 3-phase profiles, displayed as speed vs. position along path.

becomes under-defined, so Δt_2 and Δt_5 each have two solutions: one negative and one positive. The positive solution is chosen. In the case where both solutions are negative, accel-jerk tuning is attempted. The 6-phase profile can be expressed as $(j_1 > 0, j_2 = 0, j_3 < 0, j_4 < 0, j_5 = 0, j_6 > 0)$ Should that yield no positive time intervals, one of the two 4-phase profiles then becomes appropriate, since L is not large enough to accommodate accelerating past the end speed. If $v_i > v_f$, then a 4-phase profile is attempted. However, if $v_i > v_f$, then a reversed 4-phase profile is attempted.

3) 4-phase: When $v_i < v_m \simeq v_f$, the vehicle should increase speed, then maintain that speed for the distance remaining in the segment. The 4-phase profile can be expressed as $(j_1 > 0, j_2 = 0, j_3 < 0, j_4 = 0)$

4) Reversed 4-phase: When $v_i \simeq v_m > v_f$, it would be nonsensical to immediately brake. The initial speed is maintained for a period before applying the brakes. The reversed 4-phase profile can be expressed as $(j_1 = 0, j_2 < 0, j_3 = 0, j_4 > 0)$

5) 3-phase: A change from one speed to another. All of the above profiles are composed of some combination of a 3 phase profile and a $j = 0$ phase. This presents a difficulty in that either L or v_f may be enforced, but not both (there is no solution to enforce both). As such, this profile is chosen only for a segment in which it is impossible to completely arrive at the end speed given the initial conditions. When these discrepancies in the feasible v_f for a segment arise, the corresponding value in v_p is updated so that the new conditions may be accounted for in the subsequent segment. The 3-phase profile can be expressed as $(j_1 \pm 0, j_2 = 0, j_3 \mp 0)$

For each of the profiles, it is possible that the chosen values of $\mathbf{a}_m$ and $\mathbf{j}_m$ is not feasible, resulting in a negative time interval for one or more phases. In many cases, modifying $\mathbf{a}_m$ and $\mathbf{j}_m$ such that they still lie within $\mathbf{a}_m^{max}$ and $\mathbf{j}_m^{max}$ will remedy this. Specifically, jerk may be increased while peak acceleration is decreased. This process is referred to herein as *accel-jerk tuning*. If no acceptable values are found within the valid ranges of $\mathbf{a}_m^{max}$ and $\mathbf{j}_m^{max}$, a different jerk profile must be chosen.

To decide which profile to employ, a cascade approach is used. Analytic values of time intervals are evaluated for each profile by plugging in new values of $\mathbf{a}_m$ and $\mathbf{j}_m$ until all time intervals are positive and real. The order in which they are attempted is: 7 $\rightarrow$ 6 $\rightarrow$ 4 / 4R $\rightarrow$ 3. The routines for fitting a profile to a given $\mathbf{b}, \mathbf{a}_m, \mathbf{j}_m$ to a segment using this cascade approach is referred to as a *Jerk Library*.

At this point, the subpath has been divided into segments and an analytically derived plan has been fit to each segment individually. Now a unified reference trajectory must be constructed by stitching these together. For each segment with M jerk phases, the following values are computed, which correspond to the instants at which jerk changes:

- M time durations Δt_{ref}. These must be added to the last time value from the previous segment to translate them into reference time.
- M jerk values $j(t_{ref})$, which may be appended directly to the preceding jerk values.

It is now possible to integrate $j(t)$ to determine an instantaneous trajectory at any moment in time.

C. Reactive Stop Trajectories

The procedure for planning NORMAL mode trajectories is separate from that used to compute RSTOP trajectories. For RSTOP trajectory, a 3-phase profile is used, with end conditions $v_f = 0, a_f = 0$ and initial conditions set to the current state of the navigation filter. While it would seem that the 3-phase segment solving procedure described in Sec. IV-B can be applied to reactive stopping, this is not the case. As was mentioned previously, only one of $[L, v_f]$ may be rigidly enforced for a 3-phase profile, but not both, since no closed-form solution exists for that case. Normal planning enforces L to avoid skewing segment boundaries within each subpath. However, reactive stopping requires that v_f be enforced, and that we solve for L. Reformulating a new 3-phase jerk profile accordingly is a simple matter for the symbolic solver.

Now appropriate values of a_m^- and j_m^- must be chosen to completely parameterize the trajectory. For calculating Δs_{RSTOP}, the same set of values from normal planning is used and the distance traversed in the resultant trajectory is reported as the current value of Δs_{RSTOP}. This is the procedure for computing the dynamic, speed-dependent distance required to stop at any epoch.

Once the path manager triggers RSTOP, the trajectory is then recomputed with increasingly higher absolute values of a_m^- and j_m^- until $\Delta s_{RSTOP} \leqslant \Delta s_{avail}$. Should one of the values a_m^- or j_m^- required to fulfill this condition exceed the limits $a_m^{-,max}$ or $j_m^{-,max}$, then limit braking parameters are set such that $a_m^- = a_m^{-,max}$ and $j_m^- = j_m^{-,max}$, and the user is alerted.

The path must now be truncated for mapping back to Cartesian coordinates. Within the trajectory tracker, all portions of the path leading up to the location of RSTOP triggering are removed, as well as the portions after the location which corresponds to $s_{veh} + \Delta s_{RSTOP}$. The vehicle now begins sampling from this trajectory and executes a smooth reactive stop.

The time at which the trajectory plan must be referenced is not always known. The solution to this problem is known as trajectory tracking, and is discussed in the next section.

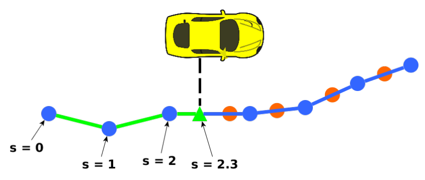


Fig. 11. Showing projection of the current position of the car shown by green triangle to s coordinate system. Each circle represents a node, and the lines between them are links. The orange dots indicate future trajectory samples which are sent to the controller.

V. Trajectory Tracking

After planning has been performed, a reference signal must be regularly supplied to the control system in order for the vehicle to carry out the plan. The primary difficulty here is that the trajectory plan has been developed in the s-frame using a reference time which has an unknown drift relative to real time. As such, we must find a way to look up the coordinates of the vehicle within the reference trajectory using information typically available for autonomous cars. A minimal set of such information includes position, velocity vector, heading, and acceleration in the body frame. It is assumed that the vehicle controller will need samples of a small window into the future trajectory to pilot the vehicle.

The general idea is to project the vehicle's current position onto the path, and then find the cumulative distance along the path between the start point and the projection point. The Cartesian vehicle position is thus mapped onto the path at a Frenet position which corresponds to the vehicle's current reference time. Once this reference time is obtained, it may be used to look up position, speed, acceleration, and jerk along the curve of the path over the future time window. From there it is a matter of simple geometry to transform these values into the coordinate system of the path and, subsequently, any other coordinate system which may be needed.

Several algorithms exist to find the optimal projection of the target vehicle's present kinematic state (position, velocity and acceleration vectors) onto the path. Such procedures are referred to as *map matching* in the literature. A simple approximation is to perpendicularly project the vehicle's position onto a series of lines running between each pair of adjacent points in the path (this will be referred to as a *link*, and is depicted in Fig. 11). Projections which lie outside of the link's two constituent path points are snapped to the nearest one. The projection which is closest to the vehicle's actual position is then assumed to be the correct path projection $\mathbf{r}_p$. Any number of modifications can be made using heading, velocity vectors, and previous information to allow this method to be extended to complex overlapping paths or self-intersections.

Choice of an algorithm to obtain the current position of the vehicle along the length of the path s_{veh} is influenced by the form of the path plan. We assume that the path is linear between waypoints to significantly reduce the computational effort. So s_{veh} at any epoch is the sum of the link

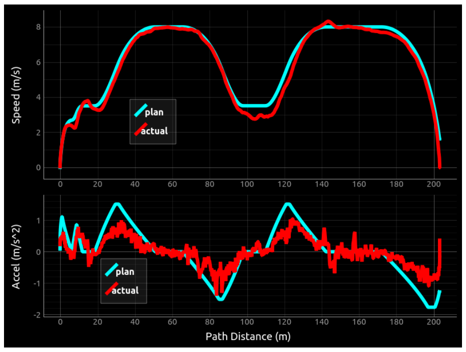


Fig. 12. Planned speed and longitudinal acceleration vs. measured values

lengths behind its perpendicular projection $\mathbf{r}_p$. However, the error in this approximation approaches zero as the spacing between path points approaches zero. Other, more accurate, approximations which work well include cubic splines, arc splines, and Bezier curves, which have solutions for closest point projection and arc-length calculation [24]–[26].

Now the reference time corresponding to the vehicle's location t_{veh} is needed to look up $v_x(t_{veh})$, $a_x(t_{veh})$, and $j_x(t_{veh})$. Additionally, it is also needed to sample $s(t)$ at intervals ahead of the vehicle. Since it is known that $s(t)$ is piecewise cubic, reference time as a function of vehicle position along the path, $t(s)$, may be approximated as cubic using a spline. To get the knots, $s(t)$ is sampled at $\Delta t = 0.01sec$. This yields two vectors s' and t' which are then used to create a univariate cubic $t(s)$ spline. Evaluating $t(s_{veh})$ then yields the current reference time.

To summarize, in the last three sections we described breaking an urban path into sub-paths. We explained the computation of velocity and jerk profile for segments within sub-paths that honor constraints for speed, velocity and acceleration as well as reactively slow down or stop for pedestrians on the urban path. We then described how our trajectory plan is supplied regularly in real-time to the autonomous car controller.

Fig. 12 shows planned speed and acceleration for the second sub-path between the two stop signs in Fig 5 (highlighted in green). Overlayed on top are measured values of both quantities for a single autonomous driving run, both of which track quite well (the RMS speed error is $0.55\ m/s$, though this is largely dependent upon the controller) . Note that they are not smoothed or bias-corrected, thus the acceleration is quite noisy. Relatively high jerk and acceleration are used to compensate for significant damping effects and time constants in the controller. The controller used in this work was originally tuned for interstate driving, so its behavior at low urban speeds has not been optimized leading to significant steady state acceleration errors.

VI. Summary

In this paper, we have presented a novel real-time method for planning longitudinal trajectories in an autonomous car to follow an urban path with online updates to avoid pedestrians on the roadway by slowing down or stopping. We built a robust stopping model as part of Trajectory Planner reliably integrated with the Path Manager. We can travel along a desired path, stop at stop signs, stop for pedestrians in the roadway, and continue after they leave. Our system gracefully handles complex cases that were not explicitly programmed, such as follow pedestrians if they continue walking on the road away from the car. Our system works with multiple pedestrians as we only react to the closest pedestrian ahead of us.

We make three contributions in this paper. First, we present an integrated trajectory planner that simultaneously limits jerk, velocity and acceleration to pre-set desired values while being responsive to the presence of pedestrians. Second, our trajectory planner has closed form solution and is suitable for online implementation on a car, unlike systems requiring solutions to nonlinear optimizations. Finally, we have confirm the efficiency and reliability of our trajectory planner on a test vehicle with over 100 hours of testing under fully autonomous driving conditions on urban roads with pedestrians in random locations.

In future, we would like to handle other roadway traffic, perceiving and reacting to other vehicles. We would also like to perceive and stop for stop signs and red-lights, rather than using map information.

References

[1] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for bertha a local, continuous method," in *IEEE Intelligent Vehicles Symposium*, June 2014.

[2] M. Bahram, , A. Wolf, , M. Aeberhard, , and D. Wollherr, "A prediction-based reactive driving strategy for highly automated driving function on freeways," in *Intelligent Vehicles Symposium Proceedings*, Dearborn, Michigan, USA, 2014, pp. 400–406.

[3] W. Xu and J. M. Dolan, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Proceedings of the International Conference on Robotics and Automation*, May 2012, pp. 2061–2067.

[4] A. Chebly, G. Tagne, R. Talj, and A. Charara, "Local trajectory planning and tracking for autonomous vehicle navigation using clothoid tentacles method," in *International IEEE Conference on Intelligent Vehicles Symposium (IV)*, Seoul, South Korea, Jun 2015.

[5] C. Guarino Lo Bianco, A. Plazzi, and M. Romano, "Velocity planning for autonomous vehicles," in *IEEE Intelligent Vehicles Symposium, 2004*, no. 1. IEEE, 2004, pp. 413–418.

[6] C. Guarino Lo Bianco and M. Romano, "Bounded velocity planning for autonomous vehicles," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, no. 2. IEEE, 2005, pp. 685–690.

[7] C. G. L. Bianco and M. Romano, "Optimal velocity planning for autonomous vehicles considering curvature constraints," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, no. April. IEEE, apr 2007, pp. 2706–2711.

[8] C. G. L. Bianco, "Kinematically constrained smooth real-time velocity planning for robotics applications," in *2009 IEEE International Conference on Control and Automation*, no. 2. IEEE, dec 2009, pp. 373–378.

[9] C. Guarino Lo Bianco, "Minimum-Jerk Velocity Planning for Mobile Robot Applications," *IEEE Transactions on Robotics*, vol. 29, no. 5, pp. 1317–1326, oct 2013.

[10] J. P. Rastelli, R. Lattarulo, and F. Nashashibi, "Dynamic trajectory generation using continuous-curvature algorithms for door to door assistance vehicles," in *2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, June 8-11, 2014*, 2014, pp. 510–515.

[11] X. Li, Z. Sun, Z. He, Q. Zhu, and D. Liu, "A practical trajectory planning framework for autonomous ground vehicles driving in urban environments," in *2015 IEEE Intelligent Vehicles Symposium, IV 2015, Seoul, South Korea, June 28 - July 1, 2015*, 2015, pp. 1160–1166.

[12] C. Pradalier, J. Hermosillo, C. Koike, C. Braillon, P. Bessire, and C. Laugier, "The cycab: a car-like robot navigating autonomously and safely among pedestrians," *Robotics and Autonomous Systems*, pp. 51–68, 2005.

[13] R. Benenson, P. Paris, T. Fraichard, and M. Parent, "Integrating perception and planning for autonomous navigation of urban vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2006.

[14] T. Gu and J. M. Dolan, "Toward human like motion planning in urban environments," in *Intelligent Vehicles Symposium Proceedings*, Dearborn, Michigan, USA, 2014, pp. 350–355.

[15] A. Mgelmose, M. M. Trivedi, and T. B. Moeslund, "Trajectory analysis and prediction for improved pedestrian safety: Integrated framework and evaluations." in *Intelligent Vehicles Symposium*. IEEE, 2015, pp. 330–335.

[16] J. Johnson and K. K. Hauser, "Optimal longitudinal control planning with moving obstacles," in *2013 IEEE Intelligent Vehicles Symposium (IV), Gold Coast City, Australia, June 23-26, 2013*, 2013, pp. 605–611.

[17] J. Villagra, V. Milanés, J. Pérez, and J. Godoy, "Smooth path and speed planning for an automated public transport vehicle," *Robotics and Autonomous Systems*, vol. 60, no. 2, pp. 252–265, 2012.

[18] J. Villagra, V. Milanes, J. Perez, J. Godoy, and E. Onieva, "Path and speed planning for smooth autonomous navigation," in *Intelligent Vehicles Symposium Proceedings*, Madrid, Spain, 2012.

[19] A. Gepperth, M. Garcia Ortiz, and B. Heisele, "Real-time pedestrian detection and pose classification on a GPU," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, oct 2013, pp. 348–353.

[20] A. Gepperth, E. Sattarov, B. Heisele, and S. A. R. Flores, "Robust visual pedestrian detection by tight coupling to tracking," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, oct 2014, pp. 1935–1940.

[21] A. K. Maurya and P. S. Bokare, "Study of Decleration Behaviour of Different Vehicle Types," *International Journal for Traffic and Transport Engineering*, vol. 2, no. 3, pp. 253–270, sep 2012.

[22] L. L. Hoberock, "A Survey of Longitudinal Acceleration Comfort Studies in Ground Transportation Vehicles," *Journal of Dynamic Systems, Measurement, and Control*, vol. 99, no. 2, p. 76, 1977.

[23] G. Long, "Acceleration Characteristics of Starting Vehicles," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1737, no. 00, pp. 58–70, jan 2000.

[24] H. Wang, J. Kearney, and K. Atkinson, "Robust and efficient computation of the closest point on a spline curve," in *In Proceedings of the 5th International Conference on Curves and Surfaces*, 2002, pp. 397–406.

[25] ——, "Arc-Length Parameterized Spline Curves for Real-Time Simulation," *Curve and surface design: Saint-Malo 2002.*, pp. 387–396, 2003.

[26] A. Schindler, G. Maier, and S. Pangerl, "Exploiting arc splines for digital maps," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, oct 2011, pp. 1–6.