Application Report

Dynamic Backup Lines

Jonathan Key, Shashank Dabral, Lucas Weaver

ABSTRACT

This application report describes an algorithm to generate dynamic, animated, rearview lines based on steering wheel input, using the cost-optimized, GPU-less, TDA3x platform, leveraging one of its C66x DSP cores.

Table of Contents

List of Figures

Trademarks

1 Ackermann Steering Curve Generation and Drawing Algorithm

The Ackermann steering model is commonly used throughout the automobile industry, to mathematically model the path that the tires of a car follow during a turn. As Figure 1-1 shows, the four tires follow a set of four concentric circles.

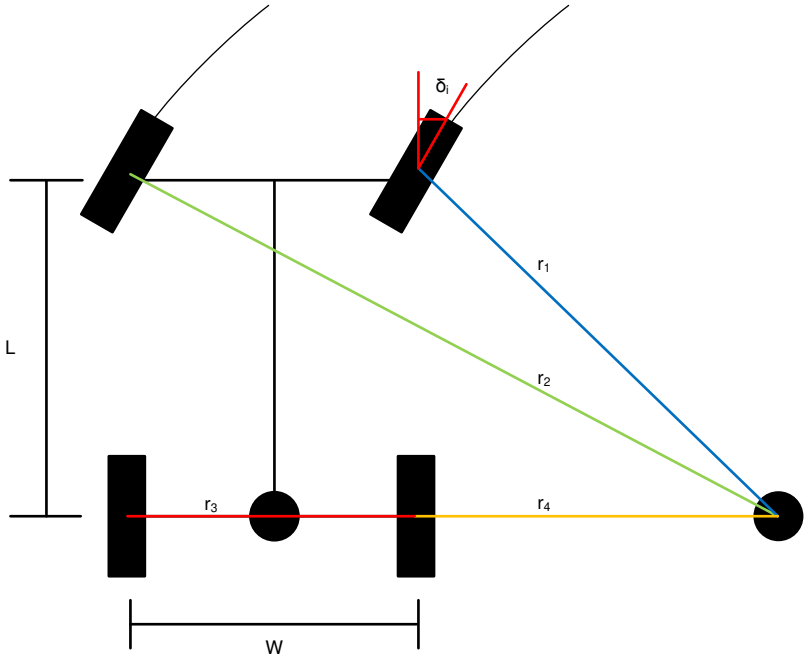

Figure 1-1. Ackermann Steering Curves

Given the wheelbase (L), tread (W), and inner tire steering angle (δ_i), the Ackermann steering model and the Pythagorean Theorem are used to determine the radius of the path of the front inner tire and the radii of the paths of the rear tires, as shown in Equation 1, Equation 2, and Equation 3. To generate the paths that the tires follow, in such a way that they can be mapped to a display, a conversion factor from inches to pixels must be applied before calculating the radii of the paths. The sample code in the Vision-SDK uses a conversion factor of 9 pixels to 1 inch.

$$r_1 = \frac{L}{\tan(\delta_i)} \tag{1}$$

$$r_4 = \sqrt{r_1^2 - L^2} \tag{2}$$

$$r_3 = r_4 + W \tag{3}$$

After finding the radii of the paths, the standard form equation of a circle can be used to generate the paths the tires follow. In the current implementation, for a given *x* value in the set xPathMin ≤ x ≤ xPathMax, the corresponding *y* value is calculated using Equation 4.

$$y = \sqrt{r^2 - x^2} \tag{4}$$

Figure 1-2 and Figure 1-3 show the MATLAB outputs of a script which takes the inner tire steering angle (δ_i) and turning direction input, and then generates rearview curves using the previous equations. Figure 1-2 shows the rearview curve for a 0° steering angle, and Figure 1-3 shows the back up curve for a 45° steering angle left turning direction input.

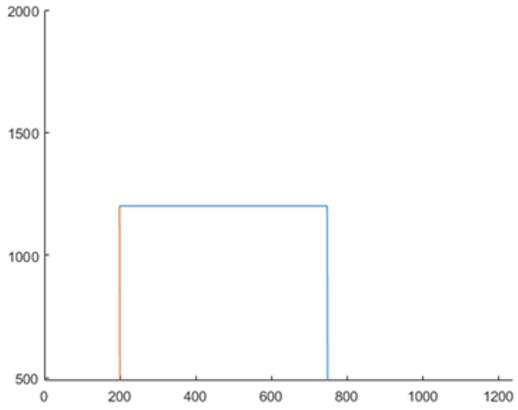

Figure 1-2. Rearview Curve for 0° Steering Angle

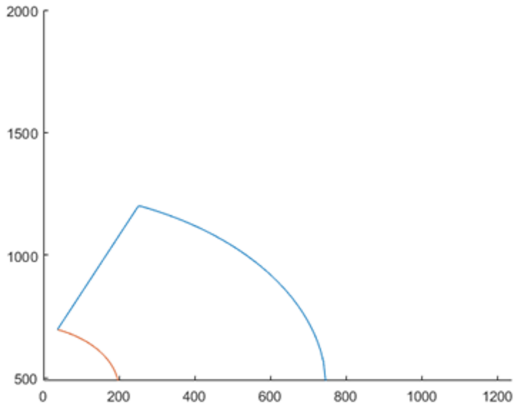

Figure 1-3. 45° Left Steering Angle

While the previous figures model the top-down view of the paths the tire follow, the rearview camera angle requires the rearview lines to appear to be converging to a finite vanishing point on the z-axis. The single-point projection transform manipulates the image such that points at infinity are mapped to a finite value in 3D space. In this case, the curves must be projected with respect to the y-axis, therefore Equation 5 was derived and used in the algorithm.

$$[X\ Y\ Z] = \left[\frac{x}{qy+1} \quad \frac{y}{qy+1} \quad 0 \right] \tag{5}$$

Before the transform can be properly applied, the center point of the rear axle is mapped to the origin. Specifically, the curves must be translated so that the path of the driver-side tires is in the second quadrant, and the path of the passenger-side tires is still the tread length away from the driver side tire. Figure 1-4 shows the MATLAB output of the previous transform applied to the curve generated in Figure 1-3.

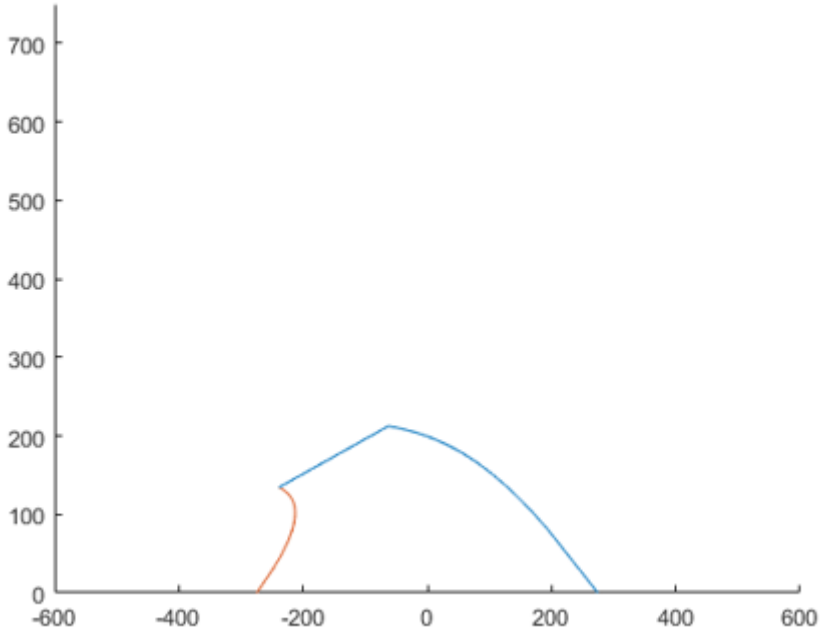

Figure 1-4. Projected Rearview Lines From Figure 1-3

Lastly, the curves are translated so that they are in the rearview camera viewing window. Figure 1-5 shows the display output of the TDA3X RVP running the surround view + rearview use case, with the dynamic rearview lines being drawn using the draw2D API included in the Vision-SDK.

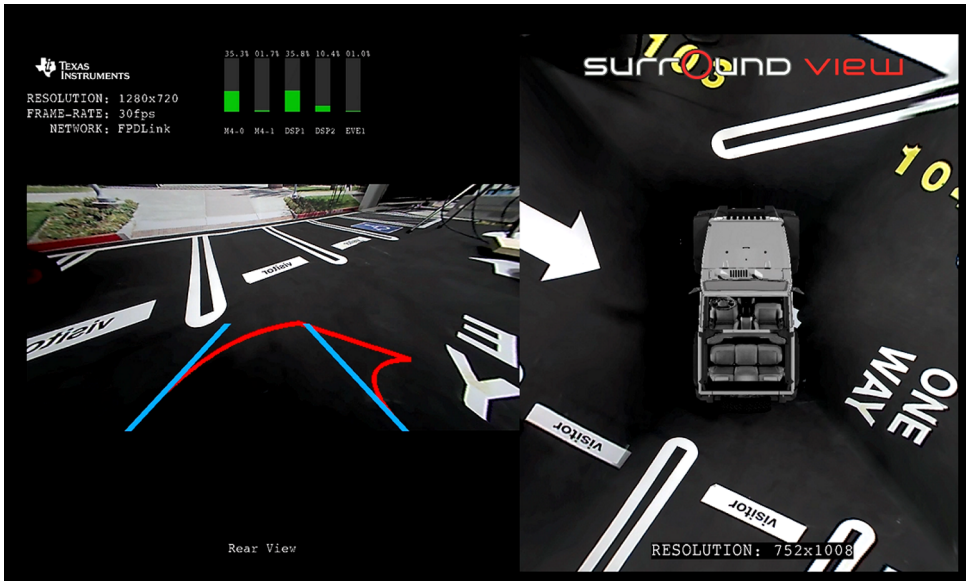

Figure 1-5. Display Output of Algorithm Running on TDA3x

2 DSP Link Generation and Integration With Vision_SDK

The Vision-SDK, from TI, contains a use-case generation tool which simplifies development by letting the developer quickly specify which algorithms run on which core, and in what order. Using the use-case generation tool, a new link was added to an existing use-case that displays 3D surround view and the rearview camera. The algorithm described in the previous section was implemented on the C66x DSP in the Alg_drawRearview link. Figure 2-1 shows the use-case flow chart generated from the output of the use-case generation tool. Additional documentation on the link and chain structures and use-case generation tool is in vision_sdk/docs.

The AlgorithmLink_drawRearviewCreate function is called when the use-case is instantiated. The necessary link parameters for the links and chains framework are generated with this function. This includes allocating the memory and initializing the input and output queues, setting up the output color format, the required buffer information for the draw2D API, and the data structures used within the algorithm link.

Next, the link architecture invokes the AlgorithmLink_drawRearviewProcess function using a notification generated by the previous link, indicating that the input queue is ready to be processed. The use-case architecture requires the rearview lines to be drawn at the same frame rate that is output to the display. However, the current implementation of the Alg_drawRearview link calculates the discrete points on the rearview curves at a rate of 15 frames per second (FPS) and stores them in a buffer that is drawn at a rate of 30 FPS.

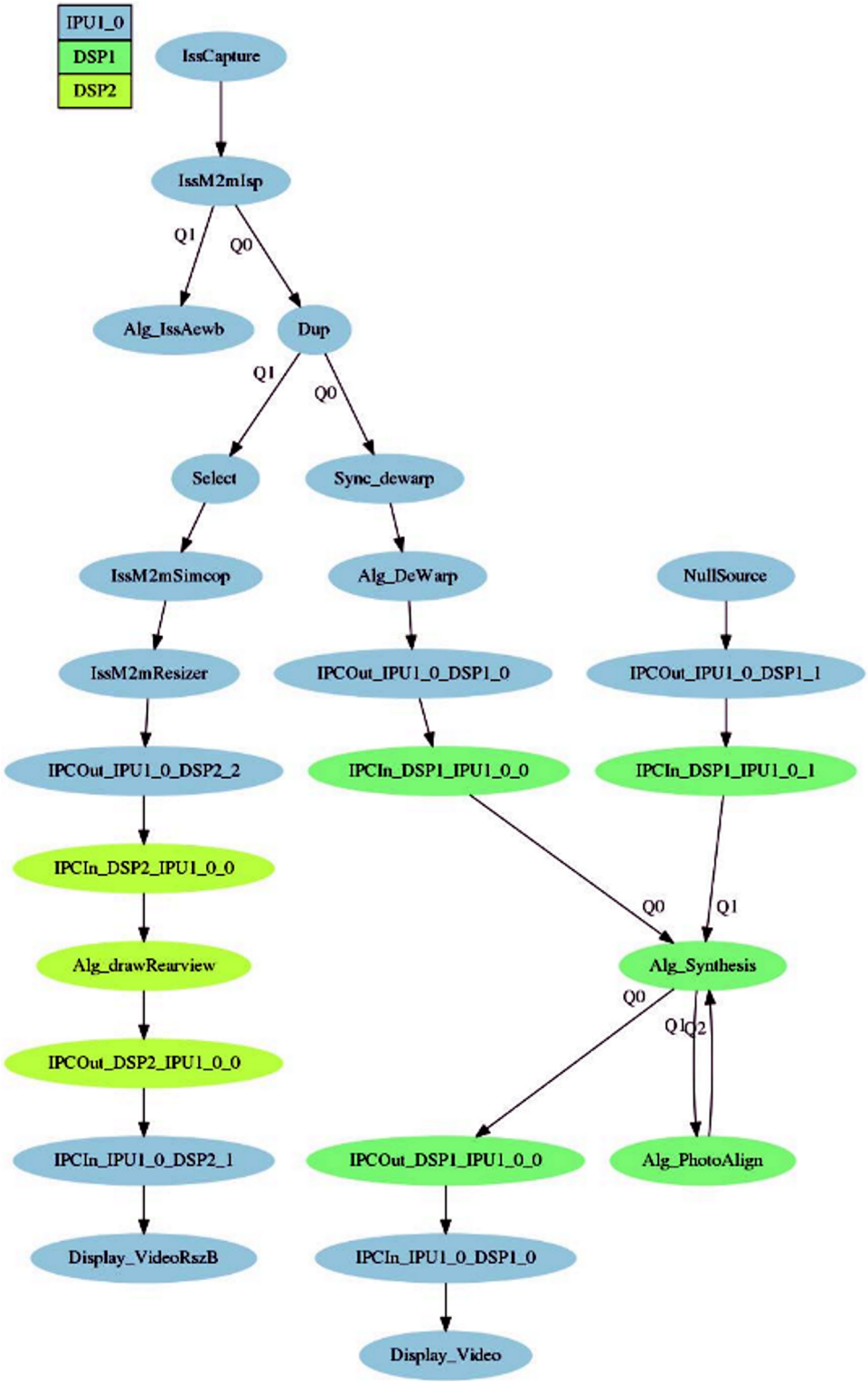

Figure 2-1. Dynamic Rearview Lines + 3D Surround View Use-Case Links and Chains Flow Chart

The Alg_drawRearview function leverages the draw2D API to draw the rearview lines on the video frame. This occurs within the AlgorithmLink_drawAckermannSteering function, where the discrete points of the rearview lines are calculated and drawn, based on the steering direction and inner wheel steering angle. The function stores the coordinate pairs of each discrete point in a set of buffers, and then calls the Draw2D_drawLine function to draw a line segment between two adjacent points. Draw2D_drawLine has a line parameter struct input that lets the developer set the color format, color, and thickness (in pixels) of the lines drawn.

Currently, the dynamic rearview lines demo in the Vision-SDK is a proof of concept that uses a switch statement to update the steering angle and steering direction each time the AlgorithmLink_drawRearviewProcess function is invoked. In an application where a steering wheel is available, the developer could implement a CAN BUS monitor link and a data sync link that could aggregate the steering wheel input data with the video frame that has the same timestamp into the input queue of the Alg_drawRearview link. The data could then be parsed in the AlgorithmLink_drawRearviewProcess function and the AlgorithmLink_drawAckermannSteering function could be called to generate and draw the rearview lines.

3 Loading

This application is the only algorithm running on DSP2 and is 9% to 12% load at 500-MHz clock speed. Surround view is running on DSP1, approximately 35% at 500-MHz clock speed. As previously mentioned, the discrete points are calculated at a rate of 15 FPS, while the lines are drawn at the full 30 FPS to reduce the DSP loading of the algorithm. Further decreasing the rate at which the points are calculated should continue to decrease the DSP load. Another approach to reduce the load is to decrease the thickness of the lines drawn in by the Draw2D_drawLine function by modifying the LinePrm.lineSize. The final recommended step that can be taken to reduce load is to modify the DISCRETE_RES parameter in drawRearviewLink_priv.h, decreasing this value decreases the number of points generated for each curve, and improves loading performance while sacrificing visual quality.

4 Function Inputs

```
 /*******************************************************   Include
Files*************************************************************/#include
"drawRearviewLink_priv.h"#include <include/link_api/system_common.h>#include <src/rtos/utils_common/
include/utils_mem.h>#define L  (110.2*9)//Wheelbase: 110.2"#define W  (61.0*9) //Rear wheel tread:
61.0"#define W2 (W/2.0)
```

In the drawRearviewLink_DynamicLines.c file, the L macro corresponds to the wheelbase of the car in inches, and W corresponds to the rear wheel tread. As mentioned previously, a conversion factor of 9 pixels to 1 inch is used in this demo.

```
void AlgorithmLink_drawAckermannSteering(AlgorithmLink_drawRearviewObj *pObj)
{
 AlgorithmLink_RearviewOverlayDrawObj *pRearViewOverlayDrawObj;
 pRearViewOverlayDrawObj = &pObj->rearViewOverlayDrawObj;
// Compile Time Inputs
Int32 x0Driver = 200;
Int32 x0Pass = x0Driver + W;
Int32 y0 = Y0;
float q = 0.003378;
```

In the AlgorithmLink_drawAckermannSteering function in the same file, the variable q controls the point at which the perspective transform converges.

```
/*********************************************** Data
structures***********************************************/typedef struct {
    Bool isFirstTime; /* Are we invoking the draw API for this first time */
    Bool displayTopView;
    UInt32 state;
    UInt32 prevState;
    UInt32 delay;
    Int32 aLong;            // Variable for ellipse steering model
    Int32 steeringAngle;
    Bool turningDir;      // TRUE = Right Turn; FALSE = Left Turn
    Bool drawBuf;
    /* If TRUE will draw from Buffer arrays below, else generate points, fill buffer, then draw*/
short iDriver;
    short iPass;
    Int32 xDriverBuf[DISCRETE_RES];
    Int32 yDriverBuf[DISCRETE_RES];
    Int32 xPassBuf[DISCRETE_RES];
    Int32 yPassBuf[DISCRETE_RES];
} AlgorithmLink_RearviewOverlayDrawObj;
```

In the drawRearview_priv.h file, the structs AlgorithmLink_drawRearviewObj and its member AlgorithmLink_RearviewOverlayDrawObj (shown previously) are declared. The steeringAngle and turningDir members of AlgorithmLink_RearviewOverlayDrawObj should be updated by input from the CAN bus.

5 References

- vision_sdk/docs
- Zhao, Jing-Shan and Feng Xiang Liu, Zhi-Jing and Dai, Jian. (2013). *Design of an Ackermann Type Steering Mechanism*. Journal of Mechanical Engineering Science.
- Paul Heckbert, *Fundamentals of Texture Mapping and Image Warping* (pages 17-21), Master's thesis, UCB/CSD 89/516, CS Division, U.C. Berkeley, June 1989

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265