Human vs. Computer Slot Car Racing using an Event and Frame-Based DAVIS Vision Sensor

T. Delbruck[1], M. Pfeiffer[1], R. Juston[2], G. Orchard[3], E. Müggler[4], A. Linares-Barranco[5], M.W. Tilden[6]

1: Institute of Neuroinformatics, Univ. of Zurich and ETH Zurich, Switzerland,
2: Biorobotics Team, Institute of Movement Sciences, CNRS / Aix-Marseille Univ.
3: Singapore Institute for Neurotechnology (SINAPSE)
4: Robotics and Perception Group, Univ. of Zurich
5: Robotics and Technology of Computers Lab, Univ. of Seville
6: Design consultant, Wowwee Ltd, Hong Kong

Abstract –**This paper describes an open-source implementation of an event-based dynamic and active pixel vision sensor (DAVIS) for racing human vs. computer on a slot car track. The DAVIS is mounted in "eye-of-god" view. The DAVIS image frames are only used for setup and are subsequently turned off because they are not needed. The dynamic vision sensor (DVS) events are then used to track both the human and computer controlled cars. The precise control of throttle and braking afforded by the low latency of the sensor output enables consistent out-performance of human drivers at a laptop CPU load of <3% and update rate of 666Hz. The sparse output of the DVS event stream results in a data rate that is about 1000 times smaller than from a frame-based camera with the same resolution and update rate. The scaled average lap speed of the 1/64 scale cars is about 450km/h which is twice as fast as the fastest Formula 1 lap speed. A feedback-controller mode allows competitive racing by slowing the computer controlled car when it is ahead of the human. In tests of human vs. computer racing the computer still won more than 80% of the races.**

I. Introduction

The DAVIS is a neuromorphic camera that outputs static image frames concurrently with dynamic vision sensor (**DVS**) temporal contrast events [1][2]. DVS address-events (**AEs**) asynchronously signal changes of log intensity. The AE timestamp (in microseconds) codes the time of the events. Pixels with DVS output are neuromorphic abstractions of retinal ganglion cells in biological retinas. Their sub-ms latency, sparse output, and kHz pixel bandwidth has led to applications requiring high speed object tracking with short-latency feedback, e.g. [3][4]. In this work, we use the DVS outputs to track slot cars and control one of the cars to race competitively against human drivers. In this application, the DAVIS static frames were useful for setting up the sensor and adjusting focusing, but subsequently the frames were not needed and were turned off.

II. Hardware and Software Setup

Fig. 1 shows a racetrack together with sample DVS data produced by a single car driving around the track. The 240x180 pixel DAVIS was mounted in "eye of god" view over the table using a wide angle 2.6mm lens with a horizontal field of view of 81° to cover the slot car track. As the cars go around the track, they create DVS events, which are shown superimposed as 3D space-time events over the photo of the track. These events are used as described later to track the cars and to control the computer car throttle and braking. The events captured from the DAVIS are transmitted to the host PC over a USB interface. Individual events are time-stamped with 1us resolution.

Slot cars contain a DC motor, a pin to guide the car along the slot in the track, two brass brushes that pick up power from the metal rails of the track, and a magnet that helps hold the car onto the steel track rails. Power to the car is normally regulated by a simple throttle controller consisting of a wire-wound resistor. Racers attempt to go as fast as possible around the track without flying off it. We used a HO-scale (1/64) system from AFX Racing

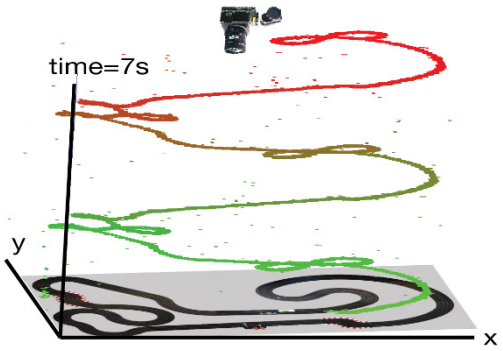

Fig. 1. A track layout with the stream of DVS events (dots) caused by a moving car shown as dots in 3D space-time. The average event rate caused by a moving car is about 5k events/sec.

(www.afxracing.com) with car chassis type SRT. The cars are 7cm in length, and the track (**Fig. 2**) had 15 turns in a length of 805cm, or 900 pixels on the sensor image. The fastest lap times are about 4.1s. The slot car speed of 200cm/s scaled to Formula 1 size is 450km/h; for reference, the fastest average lap speed achieved in a Formula 1 race was 248km/h on a track with 11 turns (Monza 2003). Cars can accelerate to full speed in about 300ms and decelerate by friction in about the same time. The electronic motor braking (described later) slows the car even faster, allowing more aggressive driving.

Implementation

The software implementation of the slot car racer is open-sourced in the jAER project [5] in the package *ch.unizh.ini.jaer.projects.virtualslotcar* [6]. The main slot car racer class is *SlotCarRacer*. The throttle controller described in this paper is the class *HumanVsComputerThrottleController*.

1) Car Tracking, Track Model, and Track Masking

The operator sees a view of the track and other information superimposed on the DAVIS sensor output as shown in **Fig. 2**. Different parts of this display are labeled and are referred to below.

Tracking (computed by the class *CarTracker*) uses a model of the car consisting of a rectangle that is constrained to move along the track model, which is a list of track vertices in sensor pixel coordinates. A *CarCluster* is a software object based on [4] that has a 2D pixel position and a velocity along the track in track vertices per second. The track model is obtained in a semi-automated way in *TrackDefineFilter* by driving each car around alone, collecting a 2D event histogram, and then extracting a list of vertices spaced by minimum distances, starting from the peak of the

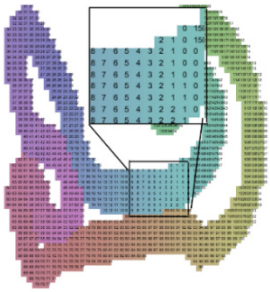

Fig. 3. Mapping from each pixel location to the nearest track vertex speeds up lookup of the nearest track vertex (number at each pixel location).

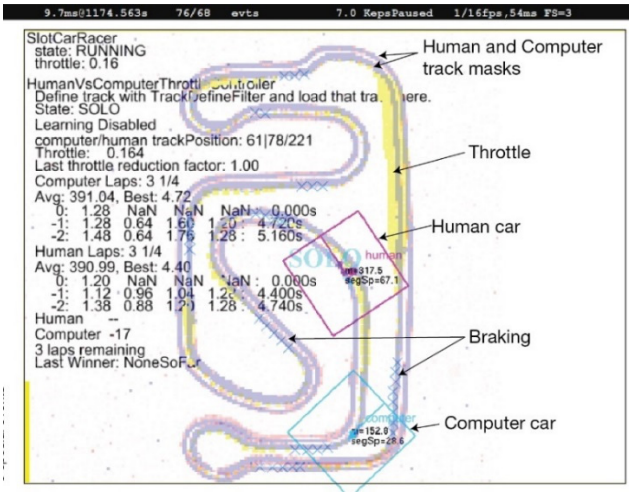

Fig. 2. Display of the slot car track with state information.

histogram. This list is then updated manually using a GUI to drag, add, and delete vertices.

Fig. 4 illustrates the car tracking update. Each DVS event input to *CarTracker* is first used to update the current car position according to the car velocity along the track, using the last update time and the current event time. This update implements the model inertia. Then the event is checked if it is near the location of the tracked car (shaded rectangle in **Fig. 4** and boundaries of boxes surrounding cars in **Fig. 2**). If so, the car model is updated by either advancing or retarding the car position along the track depending on whether the DVS event leads or lags the current car position. The amount of advancement or retarding is set by a 'mixing factor' (typically about 0.02) that mixes the DVS event position with the current car position with the mixing factor proportion. The update is done by projecting the vector v_e from current car position to the event onto the track vector v_t connecting the nearest track vertex to the next one along the track. A 2D lookup table (**Fig. 3**) maps pixel coordinates to the nearest track vertex, to speed up the search for the nearest track vertex. The car's track velocity is updated when the nearest vertex changes. The car tracker lifetime is managed by a 'mass' that decays away exponentially with time between DVS events and is incremented with each event. If the mass

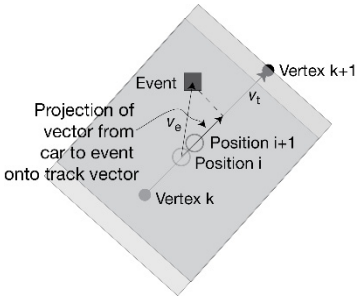

Fig. 4. CarTracker car position update.

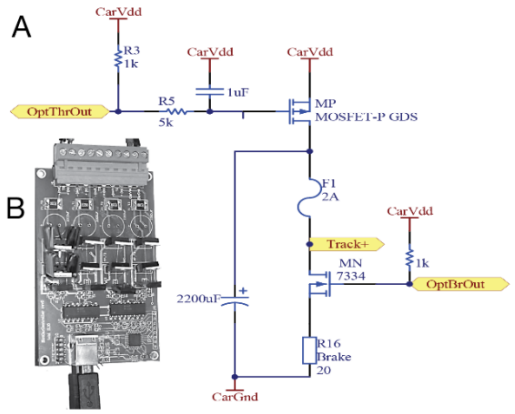

Fig. 5. The slot car controller PCB controls up to 4 lanes [7]. **A:** interface circuit of a single lane for throttle and brake control. The optocoupler pulldown outputs (OptThrOut & OutBrOut) feed the power MOSFET gates through RC low pass filters with τ=0.5ms. A fuse F1 protects against shorts. R16 is a power resistor for motor braking. **B:** Fabricated PCB with USB cable (bottom) and track/power connections (top).

falls below a certain value, the car is considered to be lost and tracking must be reinitialized.

2) *Slot Car Throttle and Braking Hardware*

We designed a slot car controller PCB (**Fig. 5**) to control the power to up to 4 slot car tracks from a computer over a full-speed USB2.0 interface. This controller controls power to the cars by 1.5kHz PWM modulation of the 17V track power, and also can short the car motor across a 20Ω resistor for electronic braking. Only one track controller is currently used and the other track is controlled by the human using the standard throttle. The controller is updated with polling interval of 1ms. The full PCB design of this controller and its firmware are available in the jAER project [7]. A prototype design did not use optocouplers and the results were frequent resets of the USB microcontroller due to large voltage transients caused by sparking of the car contact to the track which propagated back through the electronics. The final design of **Fig. 5** uses optocouplers to correct this problem.

3) *Throttle Control*

Controlling the computer car consists of setting the throttle value or applying the brake at each vertex of the track model based on a vector of throttle/brake settings called a throttle profile. An example throttle profile is shown in **Fig. 6**. We investigated a number of methods to optimize the throttle profile. Eventually we found that the fastest method is to 1) determine an initial throttle profile by settings derived automatically from the track curvature so that straight sections have higher initial throttle; 2) examining the car visually

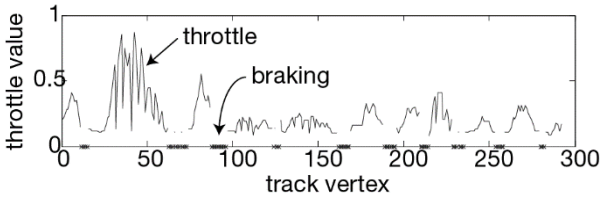

Fig. 6. Throttle and brake profile that achieves 4.1s lap times on track in Fig. 2.

and then gradually increasing the throttle or applying brake by eye, using a GUI interface to "paint" throttle and brake settings. An evolutionary method was also developed to learn the optimum throttle profile by inserting throttle increases and seeing if they result in successful laps. After a crash, the insertion is removed or braking points are inserted. The required learning time is currently still considerably longer than by manual adjustment of the profile and more work needs to be done to understand the optimum strategy.

To make racing more competitive, a mode can be enabled that slows the computer down from its optimum throttle value to a minimum value, depending on how far ahead is the computer car. Typically a value of one third of the total track for complete slow-down is effective for resulting in exciting side-by-side racing.

III. RESULTS

A series of YouTube videos document the evolution of the slot car racer since 2010 [8]–[10]. The final video shows the setup and a race between computer and human, including control that slows the computer car when it is ahead of the human.

A sample of two recorded laps by the computer controlled car is shown in **Fig. 7**. This data is taken from a different track. Over two laps, the car position increases and then wraps back to vertex 0. At the end of the straightway, motor braking rapidly decreases the car speed, as indicated by the "braking" arrow. During the last lap, motor braking is disabled ("No braking"), resulting in a crash after the straightway.

A series of ten 3-lap races between human and computer had the following results: The first human had 1 win, 2 losses, and 7 DNF (did not finish, i.e. crashed). The second human had 3 wins, 4 losses, and 3 DNFs, however after the feedback control to slow the computer car was turned off halfway through the series of ten races, the second human was no longer able to win.

Processing cost and throughput: Processing *SlotCarRacer* on a Lenovo W510 Core i7 laptop results in a CPU load of 1% to 3% for the Java virtual machine, when graphical rendering is disabled.

The update interval on the host computer was determined by instrumenting the USB data packets received by the high-priority USB processing thread

using the Java method *System.nanoTime()*. All the slot car racer processing is done in this thread, rather than the display rendering thread, which runs at most 60Hz. The DAVIS camera includes a feature called 'early packet timer' which ensures that USB FIFOs are committed to the host with intervals of at most 1.5ms, and on the host side the processing intervals closely matched this interval.

IV. Conclusion

Conventional machine vision using frame-based sensors faces a fundamental latency-power tradeoff. Low latency can only be achieved by processing at a high frame rate, which burns more power. The CPU load of less than 3% achieved in the slot car racer is a result of the low data rate averaging 5keps (thousand events per second) per car. The staring camera scenario is ideal for using the DVS, since only the small moving slot cars create DVS events. The early packet timer transmits available events from the camera at a minimum rate of 1/1.5ms=666Hz, which means that most packets sent to the host contain only about 7 events per car. This is a small amount of data to process. By comparison, if the 240x180 pixel image could be transmitted to the host at 666Hz, it would mean a data rate of 29M pixels/s, which would be a factor of about 1000 times more data.

The slot car racer robot is a popular demonstration of the use of a DAVIS sensor, mainly because racing is fun and it is a contest between human and computer that involves quick reaction times. The principle of operation is simple and easy to explain. In practice, because the computer is so precise, and because it can use motor braking, it is practically unbeatable and so to produce the illusion of a competitive race it is necessary to enable the mode where the computer car is slowed down if it is ahead.

The control of the car is in some sense open-loop because the throttle and brake are applied according to the instantaneous position of the car on the track, regardless of the car's speed. Future enhancements could focus on developing an adaptive model-based controller that regulates the speed of the car to a desired level that is safe for the curvature. Our attempts to do this were not successful because the model is surprisingly complex. The physics of car movement along the track and the action of the power applied to the car on its speed are complicated by track curvature, friction, individual car variability, motor heating, etc. However the short latency of sensor measurement could enable visual feedback on throttle control.

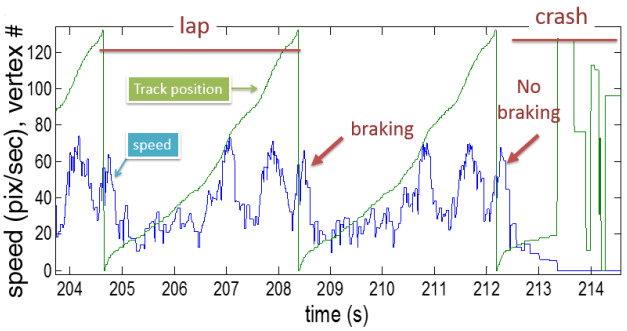

Fig. 7. Slot car position and speed vs. time on a different track. Two laps are completed successfully using motor braking to slow down just after track vertex 0. On the last lap, motor braking is disabled, resulting in a crash.

Acknowledgements

This work was supported by the Swiss NCCR Robotics, EU projects SeeBetter (FP7-ICT-270324) and Visualise (FP7-ICT-600954), Samsung and the DARPA SyNAPSE project. Prototypes were built at the 2010 Telluride Neuromorphic Cognition Engineering Workshop and the 2014 Capo Caccia Cognitive Neuromorphic Engineering Workshop. The implementation described here was demonstrated at the 2014 European Computer Vison Conference.

References

[1] P. Lichtsteiner, C. Posch, and T. Delbrück, "A 128 x 128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE J Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[2] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240x180 130 dB 3us Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE J. Solid-State Circuits*, vol. Early Access Online, 2014.

[3] A. Bolopion, Z. Ni, J. Agnus, R. Benosman, and S. Regnier, "Stable haptic feedback based on a dynamic vision sensor for microrobotics," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 3203–3208.

[4] T. Delbruck and M. Lang, "Robotic Goalie with 3ms Reaction Time at 4% CPU Load Using Event-Based Dynamic Vision Sensor," *Front. Neurosci.*, vol. 7, p. 223, Nov. 2013.

[5] jAER, *jAER Open Source Project*. 2007 [Online]. Available: http://jaerproject.org

[6] "jAER / Code / [r5522] /jAER/trunk/src/ch/unizh/ini/jaer/projects/virtualslotcar." [Online]. Available: http://sourceforge.net/p/jaer/code/HEAD/tree/jAER/trunk/src/ch/unizh/ini/jaer/projects/virtualslotcar/. [Accessed: 25-Oct-2014]

[7] "jAER / Code / [r5522] /devices/pcbs/SlotCarControllerPCB." [Online]. Available: http://sourceforge.net/p/jaer/code/HEAD/tree/devices/pcbs/SlotCarControllerPCB/. [Accessed: 25-Oct-2014]

[8] *slotCarRacingTelluride2010.wmv*. 2010 [Online]. Available: http://youtu.be/ALneVn-Ls2Q?list=UU70-pVoJ4pwIAxqpgcR7oxg. [Accessed: 06-Oct-2014]

[9] *Slot car racer controlled by DVS Capo Caccia 2014*. 2014 [Online]. Available: http://youtu.be/CnGPGiZuFRI. [Accessed: 03-Oct-2014]

[10] *Slot Car Racing with DAVIS neuromorphic vision sensor*. 2014 [Online]. Available: http://youtu.be/AsO1TWS8_VA. [Accessed: 28-Oct-2014]