ParPar : A Preliminary Application of Js-Eden

0820006

Abstract

This paper explores the current state of using Js-Eden for modelling with definitive notations. This has been achieved through modelling of the parallel parking maneuver of reverse parking a car into a gap between two currently parked cars. The process of constructing this model evaluates the current definitive notations supported by Js-Eden for constructing dependencies between observables, and for drawing to the screen, highlighting any issues which arised. It also draws attention to areas of Js-Eden which needed enhancing and extending from the base state it was supplied with to achieve the final stage of model development, and similarities and differences between Js-Eden and other tools for modelling with definitive notations.

1 Introduction

1.1 Js-Eden

Js-Eden is a tool produced by Tim Monks as part of his MSc dissertation. Js-Eden aims to "provide a system for interactive modelling with definitions, that works in the browser" (Monks, 2011). Js-Eden is produced using Javascript, which fits neatly with the structure of definitive notations. Javascript allows for event based programming, which allows code to execute when the state of element within the browser has has been altered, such as clicking on a picture, or pressing a key on the keyboard. Javascript also works natively with modern browsers, so there are no extra technologies required to get Js-Eden up and running. It also works on a local machine, not a central server, which (Monks, 2011) identifies as one of the potential limitations of WebEden[1], and as such does require a central server to process definitions.

Both Js-Eden and WebEden provide an alternative method for using and interacting with definitions constructed using the EDEN language[2]. TkEden[3] can be viewed as the current "standard" tool for developing models with the EDEN language.

Js-Eden is currently lacking in serious model development due to the short period it has been available for use. This paper produces a model of the parallel parking maneuver to evaluate whether Js-Eden is suitable for use in producing more advanced models.

[1]`http://weden.dcs.warwick.ac.uk/webeden/run/`

[2]`http://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/eden/langref/`

[3]`http://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/eden/`

1.2 Vehicle Steering

Steering of a modern vehicle is incredibly complex, and to begin to try and model all components involved is unfeasible. We restrict ourselves to producing a model of the turning of the steering wheels themselves, ignoring other factors such as the influence of the suspension and tyres on these wheels.

The majority of steering for modern vehicles is based on Ackerman Steering, which itself has been claimed to be a re-invention of a similar style of steering (King-Hele, 2002). Ackerman steering helps to solve the issue of the wheels on the outside of a turn needing to travel a further distance than the inside wheels when going around a corner. It does this by providing each wheel with independent steering, and rotating the outer wheel by a lesser degree than the inner wheel.

2 ParPar Model

The ParPar Model aims to model a four-wheeled, front-wheel steered vehicle with Ackerman steering. The model itself has been developed through three clear stages:

1. Modelling Ackerman steering of a stationary car
2. Modelling driving of a car with Ackerman steering
3. Modelling the parallel parking maneuver

2.1 Steering the car

The first iteration of the model was to simply model Ackerman steering of a stationary car. This required to construct a model of a car, similar to that found in Figure 1

(Singh, 2011) introduces a variety of formulae to calculate the angle of turn of wheel on the outside of the turn, and also how to calculate the positions of the Linkage Pivot Points at any stage of a turn. These were set up as dependencies of both the position of the car, and also the steering angle. This ensures that the state of the model remains correct, and the car does not appear to be doing anything incorrect, such as a wheel moving away from the car as the steering is increased.

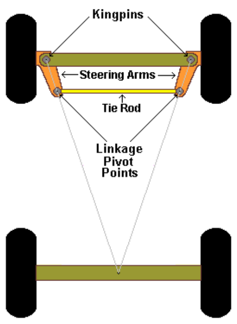


Figure 1: Ackerman steering geometry, Beam-Wiki (2011)

The steering control was modelled with a slider, which when slid right turned the wheels to the right, and slid to the left turned the wheels left. The value of the slider, to the right or left, was the steering angle of the right or left wheel respectively. This value was used in the formulae to calculate what the steering angle of the other front wheel should be, which in all cases should be less than the inner wheel, as shown in Figure 2

The wheels of the car are each constructed using a `Polygon` as a `Rectangle` does not support transformations such as rotations. To model the turning of the wheels, the `Polygon` objects were rotated about the end of the axle to which they are attached to the car with.

2.2 Driving the Car

Using another slider, it became possible to model forward, backward and stationary movement of the car. This stage of the model development also required

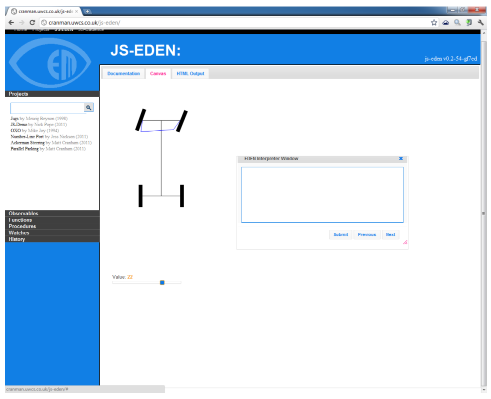

cranman.uwcs.co.uk/js-eden/
JS-EDEN:
js-eden v0.2-54-gf7ed
Documentation
Canvas
HTML Output
Projects
Jugs by Meurig Beynon (1998)
JS-Demo by Nick Pope (2011)
OXO by Mike Joy (1994)
Number-Line Port by Jess Nickson (2011)
Ackerman Steering by Matt Cranham (2011)
Parallel Parking by Matt Cranham (2011)
Observables
Functions
Procedures
Watches
History
EDEN Interpreter Window
Submit
Previous
Next
Value: 22
cranman.uwcs.co.uk/js-eden/#

Figure 2: Ackerman Steering with steering angle of 22° right

driving the car around a corner, whilst ensuring that everything currently defined from modelling a stationary car held true.

Forward and backward movement of the car was simple to model using a fixed translation of $[0, \pm 5]$, subjected to a rotation equal to the current steering angle of the car, for each fixed time frame.

Driving around a corner was again simply a rotation of the car about the centre of the turn, such that the distance travelled in one time frame was equal to the distance travelled in one time frame when driving in a straight line.

Calculating the centre of the turn is not as simple to compute. An abstraction of a car driving around a corner is that there is a point, the centre of the turn, which is also the centre of a series of concentric circles. The front wheels are both intersected by one of these circles, and also the mid-point of the rear axle. The two circles intersecting the front wheels represent the paths the front wheels would take if the car was to continue on its current trajectory, similarly so with the mid-point of the rear axle.

The distance from the mid-point of the rear axle to the centre of the turn is defined as the turning radius. The centre of the turn is always on a line extending between both rear corners of the car. Because of this, we can compute where the centre of the turn is by extrapolating the line passing through the rear corners of the car. The simplest way to achieve this is to calculate the ratio of the turning radius to rear axle length, and add this value multiplied by both the difference in x and y co-ordinates of the rear corners of the car to the rear corner co-ordinates on the side of the turn. This correctly produces the co-ordinates of where the centre of the turn can be located.

Using the co-ordinates of the centre of the turn, the model is also able to show how it is also the intersection point of lines perpendicular to the centre of all four wheels, which is shown in Figure 3.

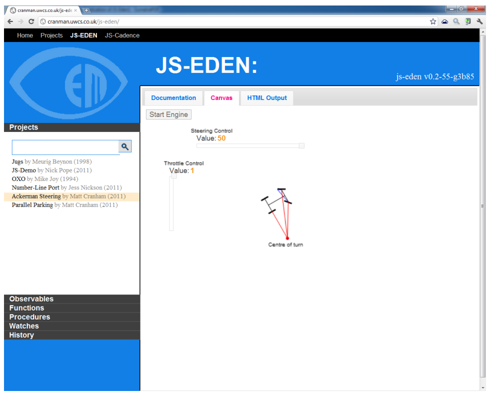

cranman.uwcs.co.uk/js-eden/
Home Projects JS-EDEN JS-Cadence
JS-EDEN:
js-eden v0.2-55-g3b85
Documentation Canvas HTML Output
Start Engine
Steering Control
Value: 50
Throttle Control
Value: 1
Centre of turn
Projects
Jugs by Meurig Beynon (1998)
JS-Demo by Nick Pope (2011)
OXO by Mike Joy (1994)
Number-Line Port by Jess Nickson (2011)
Ackerman Steering by Matt Cranham (2011)
Parallel Parking by Matt Cranham (2011)
Observables
Functions
Procedures
Watches
History

Figure 3: Driving the car

This version of the model also required use of "time" to update to positioning of the car after a fixed time frame. A `func drive` was defined which would check the value of each slider after the fixed time, which was initially set to one second, and then update the car positioning accordingly. As this was a `func` and not a `proc` there needed to be initialisation to start `drive` recursively calling itself every second, which where the "Engine" is needed. The `engine` is simply a HTML button, which when pressed starts `drive` recursively calling itself to drive the car.

2.3 Parking the Car

Parallel parking is a difficult maneuver to achieve when attempting to park a car. It requires the driver of the car to reverse into a parking space, which is parallel to the car, without causing a collision with either the two surrounding parked cars, or mounting the kerb.

The final stage of the model development is the addition of two extra stationary cars for the driven car to be parked into. Also included is a slider which controls the gap between the two stationary cars, to observe how the parallel parking maneuver is affected with a change in area to park into, see Figure 4. This iteration of the model initially used the same control mechanism as the previous version, but was replaced with arrow key controls, see Section 2.4.1.

The model has also been given "win" and "lose" conditions. A "win" has been defined such that each corner of the car must be within the `winarea`, which is

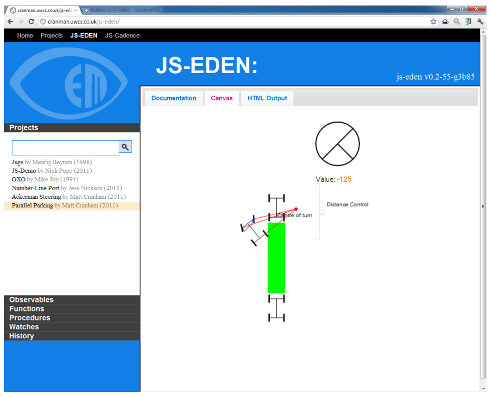
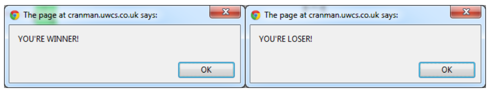

cranman.uwcs.co.uk/js-eden/
Home Projects JS-EDEN JS-Cadence
JS-EDEN:
js-eden v0.2-55-g3b85
Documentation Canvas HTML Output
Value: -125
Distance Control
Centre of turn
Projects
Jugs by Meurig Beynon (1998)
JS-Demo by Nick Pope (2011)
OXO by Mike Joy (1994)
Number-Line Port by Jess Nickson (2011)
Ackerman Steering by Matt Cranham (2011)
Parallel Parking by Matt Cranham (2011)
Observables
Functions
Procedures
Watches
History

Figure 4: Parallel parking mid-maneuver


The page at cranman.uwcs.co.uk says:

YOU'RE WINNER!

The page at cranman.uwcs.co.uk says:

YOU'RE LOSER!

(a) Win condition message (b) Lose condition message

Figure 5: Win and Lose Condition messages

a green rectangle filling the space between the two stationary cars. The "lose" condition is triggered whenever the position of a corner of the driven car is within the boundaries of one the parked cars, as this signifies a crash. Both of these messages are shown in Figure 4. These messages are presented to the used using a Javascript alert, which can be invoked in Js-Eden through use of the `${{...}}$` construct which allows arbitrary Javascript to be executed when called.

2.4 Further Refinements

2.4.1 Car Controls

It became clear that using Sliders to control the movement of the car when trying to perform a parallel parking maneuver was not ideal. The driving of the car needed much finer controls when it came to actually performing the maneuver. Javascript provides EventHandlers for the browser, which are similar to an EDEN `proc`. We are able to bind an EventHandler to any button pressed on the keyboard to then perform some function, which in this case is to control the car.

The arrow keys on the keyboard have had EventHandlers attached to them so that the Up key increases speed, the Down key decreases speed, Left key turns the wheels left, and Right key turns the wheels right. In addition to attaching these EventHandlers to the arrow keys, the model supports five level of speed forward and backwards to make controlling the car slightly more realistic.

2.4.2 Model State

It became clear that it would be possible to remove the time observable that was in use to model time progression. A state observable `state` was defined as `[car, steeringangle, throttle_value]`. The state of the model is dependent on three things, whether the position of the car has moved, whether the steering angle has been changed, or whether the throttle value has been changed. Having the steering angle and throttle value is obvious as part of the state, as when one changes, its effect is shown immediately on the car by speeding it up, slowing it down, or changing the direction of movement. The position of the car itself is also needed as part of the state to ensure that the car either continues to travel in the direction defined by the current steering and throttle values, or remains stationary.

A `proc` was set up to monitor the value of `state` so that when one of the three components of the state were changed, the car would be moved correctly in accordance with the values recorded in the state.

3 Js-Eden

JsEden has been further enhanced during the development of the parallel parking model, not only by myself but other fellow students and Nick Pope, the current maintainer for the tool.

3.1 Bug Fixing

With the initial version of JsEden provided at `http://www.dcs.warwick.ac.uk/~empublic/Js-Eden/` , there was a bug present within the list concatenation operator //. This operator should take two lists, and concatenate them together as shown in Listing 1. This behaviour occurs due to the nature of Javascript.

Listing 1: List Concatenation Examples

```
a = [1,2,3] // [4,5,6]

## Using TkEden, a == [1,2,3,4,5,6]
## Using JsEden, a == ”1,2,34,5,6”
```

As (Monks, 2011, p. 36) explains, the + operator in Javascript is overloaded so that it can process with numerical addition and string concatenation. In the example given in Listing 1, the parser for JsEden was processing `[1,2,3]` as the string `"1,2,3"` and `[4,5,6]` as `"4,5,6"`, and then concatenating these two strings to produce `"1,2,34,5,6"`.

Once the reason as to why this bug was occurring was identified, it was trivial to fix by editing the grammar for the parser so that when //was used, the parser would call the `concat()` method of the Javascript object type the lists were being translated to. This resulted in the JsEden // operator working to its defined behaviour in EDEN.

3.2 Enhancements to Js-Eden

Js-Eden was provided in a usable state in which models could be developed, however, what was, and still currently is, lacking for Js-Eden are the ability for use of other definitive notations, which TkEden allows.

(Monks, 2011, p. 48) makes the decision not to explicitly support DoNaLD syntax with Js-Eden as the DoNaLD notation is simply syntactic sugar for more complex EDEN statements. (Monks, 2011) does provide an alternative approach to directly using the DoNaLD notations, which is to use the HTML Canvas element as this provides similar features to what DoNaLD provides.

Js-Eden came provided with functions which could draw text, lines, rectangles and circles, and also allowed HTML elements such as buttons and comboboxes. What was missing from Js-Eden was the ability to manipulate these objects, through transformations such as rotation and translation, and the majority of other mathematical, trigonometric and geometric functions provided through DoNaLD .

An implementation of the Javascript Maths Object was made soon after the development of the model began as the model itself would make use of trigonometric functions to handle the steering of the car, and also the position of the car as it travels through a turn. One difference between the maths library produced, and the maths library for EDEN, is the version produced for the model uses degrees for trigonometric functions, whereas EDEN makes use of radians. This can be changed, but was chosen for clarity when analysing the model. Similarly, functions for translation, rotation, and calculating the centre of a polygon were produced.

As Js-Eden did not support the `point` construct as DoNaLD does, a list of points was simply an EDEN list of the form `[ x1,y1 , x2,y2 ...]`, and this was used for all functions and objects implemented. It became apparent quite late into the model development that `point` construct could be supported in Js-Eden , however it was decided not to implement this due to the required refactoring of a successfully working model. This could be undertaken at a later date.

Similarly, there was a need to define a new object type, `Polygon`, which took a list of these points, and draws an object to the Canvas element which visits these points in the order they are listed, returning to the first point to enclose the shape. This was done because the current `Rectangle` object does not support rotations, which the model makes use of, and also allows more complex shapes to be produced, such as pentagons, hexagons etc.

Another DoNaLD function which was required for the calculating the "win" and "lose" conditions of the final stage of the model development was the `pt_betwn_pts` function which would return a boolean value of whether the point as the second parameter was contained within the rectangle formed by the points as the first and third parameter, as the top left and bottom right corners respectively.

Due to the use of the HTML Canvas element, there were two approaches which could have been taking for implementing some of the DoNaLD functionality into Js-Eden . The approach described previously where transformations occurred on the points defining the object, and not the object itself, or the approach provided by the Canvas element. The Canvas element itself provides the ability to scale, rotate and translate anything which has been drawn to it, plus many other more advanced features not found in DoNaLD . The reason behind not choosing this method was the added complexity of linking Js-Eden and definitions within its symbol table with transformations which operate outside of Js-Eden . It would also require further understanding of how the Canvas element works to ensure that it would have been implemented correctly.

3.3 Extensions to Js-Eden

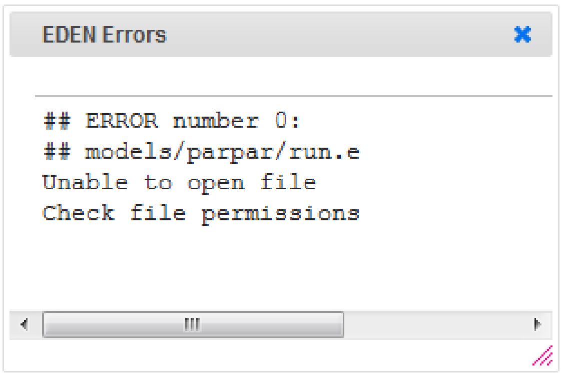
3.3.1 HTML Extensions

As mentioned previously, earlier versions of the model made use of Sliders to control the movement and steering of the car. The Sliders implemented for Js-Eden are constructed using jQuery UI[4], in a similar manner to how other HTML objects are constructed for the Canvas.

Each Slider object takes a variety of parameters to define how it behaves within a range of values, and also a name which provides a name`_value` observable which contains the current value of the Slider. This observable can then be used in any EDEN statement.

3.3.2 File Permissions

The models produced for Js-Eden can be loaded directly from the Js-Eden project list, which is a collection of models included with Js-Eden , or through the EDEN Interpreter Window. Models in the project list are loaded into Js-Eden using an Ajax request, through the `include()` method. This requires that permissions are correctly set on the files included in the model, and the directory containing them. When these are incorrectly set, the loading of the model fails and outputs an error to the Javascript console informing of the error. As the Javascript console is normally not open, this error often goes unnoticed. A modification to `eden.js` so that when the `include()` method fails in this manner, it produces an EDEN Error Message informing the user to check the file and directory permissions, as shown in Figure 6.



```
## ERROR number 0:
## models/parpar/run.e
Unable to open file
Check file permissions
```

Figure 6: Js-Eden File Permission Error Message

[4] `http://jqueryui.com/demos/slider/`

4 Conclusions

4.1 The ParPar Model

The most recent version of the ParPar Model can be found at `http://cranman.uwcs.co.uk/js-eden`, and then loading the Parallel Parking model.

I feel that the current state of the model, as described in this paper, is a good model of the parallel parking maneuver, and also a simple but accurate model of driving a car. Since the introduction of the keyboard controls, the model itself has become a lot more intuitive, and as such a better representation for driving a car.

4.2 Js-Eden

This paper has shown that Js-Eden was, and still is, in its infancy, but with more developments and models being produced, it should hopefully mature to nearer the level offered by TkEden. I have found at times the lack of functionality present in TkEden a little frustrating, but this is in no way a criticism of (Monks, 2011), as his work on Js-Eden has been pioneering in presenting Empirical Modelling in this manner.

One area where Js-Eden excels over TkEden is the observables list on the left side of the screen. This provides a way to observe the current value of definition without requirement of entering `?x;` into the TkEden interpreter. Regular expressions can be used to restrict the list of observables to only the observables which need to be observed.

I would like to see future versions Js-Eden support more definitive notations currently supported by TkEden. I understand (Monks, 2011) reasoning behind not supporting DoNaLD , as line drawings are supported through use of the HTML Canvas element, but I felt the need to produce my own versions of functions which DoNaLD provides to correctly define behaviour of the car in the model.

4.3 Acknowledgements

I would like to thank Meurig Beynon for finding the time to provide feedback and thoughts of early iterations of the development of the model, and also for valuing my opinions on future development to Js-Eden . Secondly, thanks to Tim Monks for his initial work in producing Js-Eden and providing technical support for Js-Eden during the development of the model. Finally, thanks to Nick Pope for actively maintaining Js-Eden and accepting my bug fixes and enhancements to Js-Eden .

References

Beam-Wiki. Steering Techniques, June 2011. URL `http://www.beam-wiki.org/wiki/Steering_Techniques`. 2

Desmond King-Hele. Erasmus Darwin's Improved Design for Steering Carriages–And Cars. *Notes and Records of the Royal Society of London*, 56(1): pp. 41–62, 2002. ISSN 00359149. URL `http://www.jstor.org/stable/532121`. 1

Tim Monks. A Definitive System For The Browser. Master's thesis, University of Warwick, September 2011. 1, 4, 6

Atinder Singh. Ackerman, Toe and a Few Other Random Thoughts..., June 2011. URL `http://www.scribd.com/atinders_8/d/58682411-Ackerman-Steering-Formula-Derivation`. 2