Tutor: Martin Ring
Gruppe: 1
Tobias Brandt
Stefan Heitmann

Übungsblatt 5

Aufgabenlösung
Abgabe: 22.06.2017

5.1 *Fah'n, fah'n, fah'n auf der Autobahn*

```
100 main :: IO ()
101 main = reactimate "Car" $ lift1 picToGraphic (wheels `over` carBody)

78  carBody :: Behavior Picture
79  carBody = paint green $ move position $ poly $ lift2 map (lift1 rot orientation) $ rect 0.2 0.5

81  wheels :: Behavior Picture
82  wheels = frontWheels `over` rearWheels
83
84  frontWheels :: Behavior Picture
85  frontWheels = frontWheelL `over` frontWheelR
86
87  rearWheels :: Behavior Picture
88  rearWheels = rearWheel (-0.15,0.115) `over` rearWheel (-0.15,-0.115)
89
90  frontWheelL :: Behavior Picture
91  frontWheelL = paint white $ move position $ poly $ lift2 map (lift1 rot orientation) $ lift2 map (lift1
92
93  frontWheelR :: Behavior Picture
94  frontWheelR = paint white $ move position $ poly $ lift2 map (lift1 rot orientation) $ lift2 map (lift1
95
96  rearWheel :: (Float, Float) -> Behavior Picture
97  rearWheel p = paint white $ move position $ wheelShape where
98    wheelShape = poly $ lift2 map (lift1 rot orientation) $ lift2 map (lift1 trans (constB p)) $ rect 0.03

13  trans (tx,ty) (x,y) = (x+tx, y+ty)
14
15  -- Rotate a point by an angle around the origin
16  rot :: Float-> (Float, Float)-> (Float, Float)
17  rot w (x, y)= (x* cos w- y* sin w, x* sin w+ y* cos w)
18
19  -- Convert polar coordinates to cartesian and vice versa
20  cart :: Float-> Float-> (Float, Float)
21  cart r w = rot w (r, 0) -- to cart
22  polar :: (Float, Float)-> (Float, Float)
23  polar (x, y) = (sqrt (x*x+ y*y), atan2 y x) --from cart
24
25  -- Draw a polygon
26  poly :: Behavior [(Float, Float)]-> Behavior Region
27  poly pts = shape $ lift1 Polygon pts
28
29  -- Key press events to start/stop acceleration and breaking
30  startAcc, stopAB, startBrk :: Event ()
31  startAcc = keyPress 'a'
32  stopAB   = keyPress 's'
```

```
33 startBrk = keyPress 'd'
34
35
36 acceleration :: Behavior Float
37 acceleration = 0 `untilB` ca where
38    ca = (startAcc ->> 0.5 `untilB` ca) .|.
39         (stopAB ->> 0 `untilB` ca) .|.
40         (startBrk ->> (-0.5) `untilB` ca) --.--.
41
42 rect :: Float -> Float -> Behavior [(Float, Float)]
43 rect s2 s1 =
44   let s12 = s1/2
45       s22 = s2/2
46   in constB [(-s12,-s22),(-s12,s22), (s12,s22),(s12,-s22)]
47
48
49 motionVec :: Behavior (Float, Float)
50 motionVec = lift2 cart velo orientation
51
52 position :: Behavior (Float, Float)
53 position =  vecIntegral motionVec
54
55 steerAngle :: Behavior Float
56 steerAngle = negate $ lift3 clamp (-0.8) 0.8 (fst mouse) where
57   clamp mn mx = max mn . min mx
58
59 workAngleR :: Behavior Float
60 workAngleR = 0 `untilB` ev where
61   ev = (when (steerAngle <* 0) `snapshot_` (negate $ atan (0.3 / (radius-0.1))) ->> λx -> (constB x
62         (when (steerAngle >* 0) `snapshot_` (atan (0.3 / (radius+0.1))) ->> λx -> (constB x `untilB`
63
64 workAngleL :: Behavior Float
65 workAngleL = 0 `untilB` ev where
66   ev = (when (steerAngle >* 0) `snapshot_` (atan (0.3 / (radius-0.1))) ->> λx -> (constB x `untilB`
67         (when (steerAngle <* 0) `snapshot_` (negate $ atan (0.3 / (radius+0.1))) ->> λx -> (constB x
68
69 velo :: Behavior Float
70 velo = integral acceleration
71
72 orientation :: Behavior Float
73 orientation = integral $ velo * ( steerAngle / 0.3)
74
75 radius :: Behavior Float
76 radius =  0.3 / (tan $ abs steerAngle)
```

Tests
Getestet wurden beide Aufgaben, indem die Simulation funktional ausprobiert wurde. Die Tests verliefen erfolgreich.