Fuzzy Systems for Control Applications: The Truck Backer-Upper

In this column, we introduce another tool in the advanced automation toolbox: *fuzzy logic***. The basis of fuzzy logic lies in the ambiguity in our thinking about concepts such as big, tall, slow, or bright, whose meanings are not only context dependent, but also ambiguous within a particular context. Using fuzzy sets named by these ambiguous** *linguistic variables***, we can build applications that can outperform many of their traditional counterparts. As an example of the application of this technology, we will develop a fuzzy control system that automatically backs up a truck to a specified point on a loading dock.**

James A. Freeman

Crisp Sets and Fuzzy Sets

The topic of this column is *fuzziness*. Fuzziness can be defined as the ambiguity that can be found in the definition of a concept or the meaning of a word [Terano et al. 1992]. Let me illustrate the idea with a short parable.

We are on the beach where I have made a mound with the sand. I ask you to describe the mound.

"It is a large mound of sand," you say.

"Very good," I reply, as I remove a grain of sand from the pile. "Now describe the mound."

"It is still a large mound of sand," you respond, hoping that I do not intend to remove one grain at a time for the next several years as you see me reach for another grain. Mercifully, I pick up a shovel and begin to remove sand in larger quantities.

"Tell me when the mound is no longer large," I say. You let me proceed for a minute, then stop me, saying, "Now. Now it is a small mound of sand."

"How many grains of sand would I have to add back for it to become large again?" I ask you.

"That's not a fair question," you say. "There is no exact boundary between large and medium piles of sand."

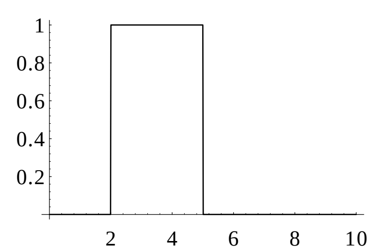
"Aha!" I exclaim. "My point exactly!" Since it is very hot on the beach, we march off in search of shade and a cold beer — you're buying, as I have just imparted valuable knowledge to you for which you are in my debt.

Think about all the ambiguous words we use regularly to describe things: big, small, tall, short, long, fast, slow, and so on. Each of those words can be used to describe sets of objects: small dogs, tall buildings. In the world of Boolean logic, boundaries between sets defined by a particular attribute are sharp. The set of *small* sand piles is distinct from the sets of *medium* and *large* sand piles. If a certain sand pile belongs to one of these sets, it cannot belong to the others. Such sets are called crisp. They have well defined boundaries, and there is no ambiguity regarding an object's membership in a crisp set.

A convenient way to represent a crisp set is by its *characteristic* function, that is, the function whose value is 1 for elements of the set and 0 outside the set. As an example, here is the characteristic function of the interval [2, 5]:

```
In[1]:=  f[x_]:= If[x >= 2 && x <= 5, 1, 0];

In[2]:=  Plot[f[x], {x, 0, 10}]
```



If we measure the size of sand piles in some units (say, kilograms or gigagrains), we could define the sets of small, medium, and large sand piles by the crisp membership functions shown in Figure 1.

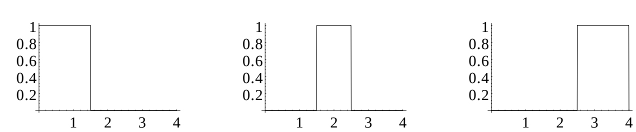


FIGURE 1. Characteristic functions for the sets of small, medium, and large sand piles. The independent variable (*x*-value) is the size of the sand pile in some chosen units. The values of the functions are always either 0 or 1.

James Freeman is an AI researcher at Loral Space Information Systems in Houston, TX, and an adjunct professor of physics and electro-optics at the University of Houston. He is coauthor of Neural Networks: Algorithms, Applications and Programming Techniques, *and author of* Simulating Neural Networks with *Mathematica, both published by Addison-Wesley.*

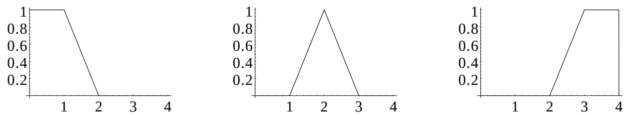


FIGURE 2. Fuzzy membership functions for the sets of small, medium, and large sand piles. The functions take values in the interval [0, 1].

A crisp set is described by a characteristic function whose value is always either 0 or 1. A *fuzzy* set is defined by a *membership* function that takes values anywhere between 0 and 1. In a fuzzy system, we might represent the sets of small, medium, and large sand piles by the fuzzy membership functions shown in Figure 2.

We say that an element "belongs" to a fuzzy set if the value of the set's membership function at that element is nonzero. The degree to which an element belongs to a particular set can be any number between zero and one inclusive. Notice that in Figure 2, certain sizes of sand pile (such as 1.5 and 2.5) belong to more than one set.

It is important to realize that we are not discussing uncertainty here; rather, class membership is equivocal in some cases. Class boundaries are often ambiguous in our minds, as in our speech; and this phenomenon is not simply a matter of our inability to be precise. None of us would likely accept $50.00 less in our paycheck even though our paycheck was close to what we normally receive; on the other hand, no one could reasonably argue that a pile of sand changed from medium size to large size on the basis of one additional grain. Yet we often impose unrealistically precise class boundaries and control setpoints in software that we write. Consider the following rules from a hypothetical control system:

If(*flow rate* is greater than 3.4
  and *flow rate* is less than 4.5)
Then(set *valve position* to 12)

If(*flow rate* is greater than 4.5
  and *flow rate* is less than 5.7)
Then(set *valve position* to 30)

Does it really make sense to adjust the valve from 12 to 30 as the flow rate varies from 4.4999 to 4.5001? In a fuzzy control system we might say something like the following:

If(*flow rate* is *very low*)
Then(set *valve position* to *open slightly*)

If(*flow rate* is *medium low*)
Then(set *valve position* to *open somewhat*)

where terms such as *very low* and *open slightly* are the names of fuzzy sets. We establish ambiguous class boundaries between *very low* and *medium low*, and between *open slightly* and *open somewhat*. Then we use fuzzy inferencing techniques to determine what action the system should take based on the measurement of the flow rate. The result is often a smoother transition between states, and a more effective control system, especially in circumstances where an accurate mathematical model of the system is difficult to determine. We will apply these ideas to an actual problem. First, however, we need to define some functions.

Constructing Fuzzy Membership Functions

The function `defineSet` takes as arguments a symbol name and a list of four points in the plane. It defines a piecewise linear function with the given name, whose graph consists of the line segments joining the four points. The implementation uses Maeder's `makeRuleConditional` function [Maeder 1991], and two auxiliary functions, `slope` and `intercept`, that compute the slope and y-intercept of a line through two given points.

```
In[3]:= MakeRuleConditional[
            f_Symbol, var_Symbol, rhs_, condition_] :=
                (f[var_] := rhs /; condition)

In[4]:= SetAttributes[MakeRuleConditional, HoldAll]

In[5]:= slope[{x1_, y1_}, {x2_, y2_}]:= (y2-y1)/(x2-x1)

In[6]:= intercept[{x1_, y1_}, {x2_, y2_}]:=(y1 x2 - y2 x1)/(x2-x1)

In[7]:= defineSet[name_, {p1_, p2_, p3_, p4_}]:=
         (ClearAll[name];
         MakeRuleConditional[name, x, 0.0, x < p1[[1]] ];
         MakeRuleConditional[name, x,
           Release[slope[p1, p2] x + intercept[p1, p2]],
           p1[[1]] <= x < p2[[1]] ];
         If[p2 != p3, MakeRuleConditional[name, x,
             Release[slope[p2, p3] x + intercept[p2, p3]],
             p2[[1]] <= x < p3[[1]] ]; ];
         If[p3 != p4, MakeRuleConditional[name, x,
             Release[slope[p3, p4] x + intercept[p3, p4]],
             p3[[1]] <= x < p4[[1]] ]; ];
         MakeRuleConditional[name, x, 0.0, x >= p4[[1]] ]; )
```

The crisp sets shown in Figure 1 are defined and plotted with the following inputs:

```
In[8]:= defineSet[crispSmall,
            {{0, 1}, {1.5, 1}, {1.5, 1}, {1.5, 1}}];
        defineSet[crispMedium,
            {{1.5, 1}, {2.5, 1}, {2.5, 1}, {2.5, 1}}];
        defineSet[crispLarge, {{2.5, 1}, {4, 1}, {4, 1}, {4, 1}}];

In[11]:= crispSets = {crispSmall, crispMedium, crispLarge};

In[12]:= Show[GraphicsArray[Plot[#[x], {x, 0, 4}]& /@ crispSets]]
```

The fuzzy sets in Figure 2 are defined and plotted similarly:

```
In[13]:= defineSet[fuzzySmall, {{0, 1}, {1, 1}, {1, 1}, {2, 0}}];
         defineSet[fuzzyMedium, {{1, 0}, {2, 1}, {2, 1}, {3, 0}}];
         defineSet[fuzzyLarge, {{2, 0}, {3, 1}, {3, 1}, {4, 1}}];

In[16]:= fuzzySets = {fuzzySmall, fuzzyMedium, fuzzyLarge};

In[17]:= Show[GraphicsArray[Plot[#[x], {x, 0, 4}]& /@ fuzzySets]]
```

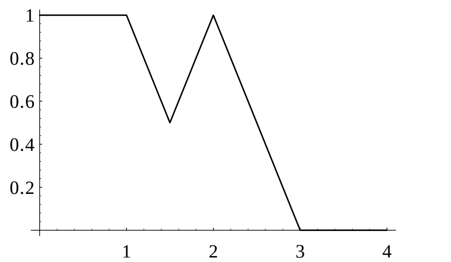
Note that in the definition of the crisp functions, the last three points in each list are the same, so the nonzero part of each graph consists of just one (nonvertical) segment. For the fuzzy functions, the middle two points are the same, so the graphs have two segments. Fuzzy membership functions typically comprise either one, two, or three segments.

We also need operations corresponding to intersection and union of sets. These operations are given by `Min` and `Max` of the membership functions:

```
In[18]:= union[functions__]:= Max[functions]
         intersection[functions__]:= Min[functions]
```

Here is an example of the union of two fuzzy sets:

```
In[20]:= Plot[union[fuzzySmall[x], fuzzyMedium[x]], {x, 0, 4}];
```



The Fuzzy Truck-Driving Academy

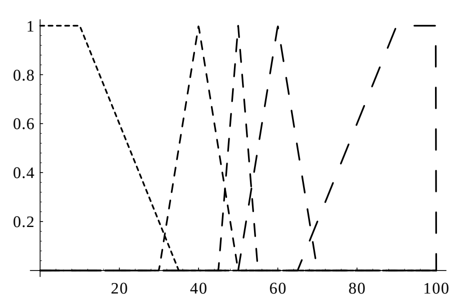
Let's apply this technology to a control problem, namely, that of backing up a truck to a specified point on a loading dock. The truck backer-upper problem has become a standard control problem. There are several varieties; the one we will work on here is a simple version taken from the work of Kong and Kosko [Kosko 1992]. Figure 3 shows a simple model. The object of the control system is to back up the truck so that it arrives perpendicular to the dock at position (x_f, y_f). The point (x, y) is the center of the rear of the truck, ϕ is the angle of the truck axis to the horizontal, and θ is the steering angle measured from the truck axis. The controller takes as input the position of the truck, specified by the pair (x, ϕ), and outputs the steering angle θ.

The linguistic variables associated with the fuzzy sets for the x position are: LE (left), LC (left center), CE (center), RC (right center), and RI (right). The following five functions define those sets:

```
In[21]:= defineSet[LE, {{0, 1}, {10, 1}, {10, 1}, {35, 0}}];
         defineSet[LC, {{30, 0}, {40, 1}, {40, 1}, {50, 0}}];
         defineSet[CE, {{45, 0}, {50, 1}, {50, 1}, {55, 0}}];
         defineSet[RC, {{50, 0}, {60, 1}, {60, 1}, {70, 0}}];
         defineSet[RI, {{65, 0}, {90, 1}, {90, 1}, {100, 1}}];
```

We can assemble the sets in a list and plot them together.

```
In[26]:= xSets[x_]:= {LE[x], LC[x], CE[x], RC[x], RI[x]};
In[27]:= Plot[Evaluate[xSets[x]], {x, 0, 100},
           PlotStyle ->
             Dashing /@ {{.01}, {.02}, {.03}, {.04}, {.05}}]
```



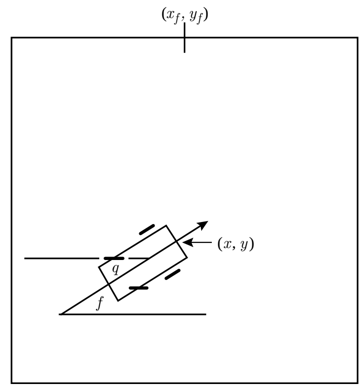
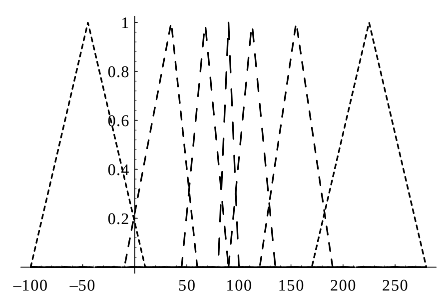


FIGURE 3. The model for the simple truck backer upper. The point (x, y) is the center of the rear of the truck, ϕ is the angle of the truck axis to the horizontal, and θ is the steering angle measured from the truck axis.

Notice that there is an odd number of sets and that the sets are concentrated near the optimum control point x_f = 50. Moreover, the values of the membership functions are at most 0.5 where the sets overlap and the sum of the values is at most 1 everywhere. These characteristics reflect rules of thumb more than any hard and fast requirements for fuzzy systems.

Fuzzy sets for the angle ϕ are: RB (right below), RU (right upper), RV (right vertical), VE (vertical), LV (left vertical), LU (left upper), and LB (left below).

```
In[28]:= defineSet[RB, {{-100, 0}, {-45, 1}, {-45, 1}, {10, 0}}];
         defineSet[RU, {{-10, 0}, {35, 1}, {35, 1}, {60, 0}}];
         defineSet[RV, {{45, 0}, {67.5, 1}, {67.5, 1}, {90, 0}}];
         defineSet[VE, {{80, 0}, {90, 1}, {90, 1}, {100, 0}}];
         defineSet[LV, {{90, 0}, {112.5, 1}, {112.5, 1}, {135, 0}}];
         defineSet[LU, {{120, 0}, {155, 1}, {155, 1}, {190, 0}}];
         defineSet[LB, {{170, 0}, {225, 1}, {225, 1}, {280, 0}}];
In[35]:= phiSets[phi_] = {RB[phi], RU[phi], RV[phi], VE[phi],
             LV[phi], LU[phi], LB[phi]};
In[36]:= Plot[Evaluate[phiSets[phi]], {phi, -100, 280},
           PlotStyle -> Dashing /@
             {{.01}, {.02}, {.03}, {.04}, {.03}, {.02}, {.01}}]
```

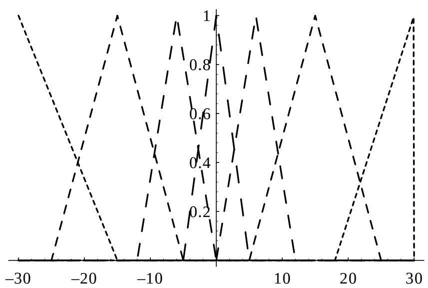


For the output variable, the sets are: NB (negative big), NM (negative medium), NS (negative small),ZE (zero), PS (positive small), PM (positive medium), and PB (positive big).

```
In[37]:= defineSet[NB, {{-30, 1}, {-15, 0}, {-15, 0}, {-15, 0}}];
         defineSet[NM, {{-25, 0}, {-15, 1}, {-15, 1}, {-5, 0}}];
         defineSet[NS, {{-12, 0}, {-6, 1}, {-6, 1}, {0, 0}}];
         defineSet[ZE, {{-5, 0}, {0, 1}, {0, 1}, {5, 0}}];
         defineSet[PS, {{0, 0}, {6, 1}, {6, 1}, {12, 0}}];
         defineSet[PM, {{5, 0}, {15, 1}, {15, 1}, {25, 0}}];
         defineSet[PB, {{18, 0}, {30, 1}, {30, 1}, {30, 1}}];

In[44]:= steerSets[theta_] =
           {NB[theta], NM[theta], NS[theta], ZE[theta],
             PS[theta], PM[theta], PB[theta]};

In[45]:= Plot[Evaluate[steerSets[theta]], {theta, -30, 30},
           PlotStyle -> Dashing /@
             {{.01}, {.02}, {.03}, {.04}, {.03}, {.02}, {.01}}]
```



With these sets we define rules which we call fuzzy associations. For example, if the angle ϕ is vertical, and the x position is right center, then we want to steer positive medium. Symbolically,

IF ϕ is VE AND x is RC, THEN θ is PM.

Rather than write out each of these rules, we can assemble them in a matrix format called a *fuzzy associative memory* (FAM). The FAM for this problem appears in Table 1.

	LE	**LC**	**CE**	**RC**	**RI**
RB	PS	PM	PM	PB	PB
RU	NS	PS	PM	PB	PB
RV	NM	NS	PS	PM	PB
VE	NM	NM	ZE	PM	PM
LV	NB	NM	NS	PS	PM
LU	NB	NB	NM	NS	PS
LB	NB	NB	NM	NM	NS

TABLE 1. Fuzzy associative memory for the truck backer-upper system. Each position in the matrix corresponds to a rule. For example, row 3, column 4 corresponds to the rule IF ϕ is RV AND x is RC, THEN θ is PM.

Given a specific input pair (ϕ, x), several rules may be applicable since the fuzzy sets overlap. The membership functions for the sets involved in each rule (such as LV, CE, and NS) are combined using AND (`Minimum`) and the result is evaluated at (ϕ, x) to get a new θ-set. The θ-sets from the different rules are then combined with OR (`Maximum`) to get the output θ-set. Figure 4 illustrates the construction of an output set from two rules, given an input (ϕ_1, x_1).

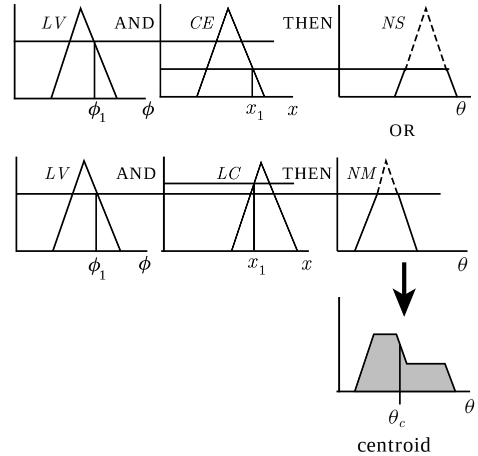


FIGURE 4. Two rules are combined to produce an output fuzzy set from an input (ϕ_1, x_1). The numerical output of the system is the centroid θ_c of the output set.

To find the actual control value, we must convert the output fuzzy set into a numerical value for θ. We call this process *defuzzification*. There are several possible methods. We will use the *centroid* method because it is the typical choice for control systems. The centroid is the "center of mass" of the region under the graph of the membership function:

$$\theta_c = \frac{\int_{-\infty}^{\infty} \theta f(\theta) d\theta}{\int_{-\infty}^{\infty} \theta d\theta}$$

Let's walk through a specific example. First, we must construct the FAM matrix:

```
In[46]:= FAM = {{PS, PM, PM, PM, PB}, {NS, PS, PM, PB, PB},
                {NM, NS, PS, PM, PB}, {NM, NM, ZE, PM, PM},
                {NB, NM, NS, PS, PM}, {NB, NB, NM, NS, PS},
                {NB, NB, NM, NM, NS}};

In[47]:= (truckFAM[theta_] = Through /@ Through @ FAM[theta]) //
           MatrixForm

Out[47]//MatrixForm=
         PS[theta]  PM[theta]  PM[theta]  PM[theta]  PB[theta]
         NS[theta]  PS[theta]  PM[theta]  PB[theta]  PB[theta]
         NM[theta]  NS[theta]  PS[theta]  PM[theta]  PB[theta]
         NM[theta]  NM[theta]  ZE[theta]  PM[theta]  PM[theta]
         NB[theta]  NM[theta]  NS[theta]  PS[theta]  PM[theta]
         NB[theta]  NB[theta]  NM[theta]  NS[theta]  PS[theta]
         NB[theta]  NB[theta]  NM[theta]  NM[theta]  NS[theta]
```

Suppose the configuration of the truck is given by ϕ = 50.0 and x = 49.0. The values of the membership functions for the ϕ and x attributes are:

```
In[48]:= phiSets[50.]
Out[48]= {0., 0.4, 0.222222, 0., 0., 0., 0.}

In[49]:= xSets[49.]
Out[49]= {0., 0.1, 0.8, 0., 0.}
```

Notice that the ϕ and x values are each members of two fuzzy sets, so this pair of inputs will result in four rule firings.

To construct the output fuzzy set, take the intersection (`Min`) of the fuzzy sets for each pair of ϕ-x attributes (such as {LV, CE}) and evaluate its membership function at the input (ϕ, x) = (50., 49.). This gives the so-called *antecedants* matrix:

```
In[50]:= Outer[Min, phiSets[50.], xSets[49.]] // MatrixForm

Out[50]//MatrixForm=
    0.    0.    0.          0.    0.
    0.    0.1   0.4         0.    0.
    0.    0.1   0.222222    0.    0.
    0.    0.    0.          0.    0.
    0.    0.    0.          0.    0.
    0.    0.    0.          0.    0.
    0.    0.    0.          0.    0.
```

The nonzero locations in the antecedants matrix correspond to the four applicable rules in the FAM matrix. We now take the intersections of the entries in the FAM matrix with the corresponding values in the antecedants matrix. The result is a matrix of θ membership functions:

```
In[51]:= MapThread[Min, {%, truckFAM[theta]}, 2] // Short[#, 3]&

Out[51]//Short=
    {{Min[0., PS[theta]], Min[0., PM[theta]],
       Min[0., PM[theta]], Min[0., PM[theta]],
       Min[0., PB[theta]]}, <<6>>}
```

Most of the entries in this matrix have the form `Min[0., _]` and so are equivalent to zero. Only those entries that correspond to applicable rules will be nonzero. The union (`Max`) of all the entries is the output fuzzy θ-set:

```
In[52]:= Max[Flatten[%]]

Out[52]= Max[{Min[0., NB[theta]], Min[0., NM[theta]],
           Min[0., NS[theta]], Min[0., PB[theta]],
           Min[0., PM[theta]], Min[0., PS[theta]],
           Min[0., ZE[theta]], Min[0.1, NS[theta]],
           Min[0.1, PS[theta]], Min[0.222222, PS[theta]],
           Min[0.4, PM[theta]]}]
```

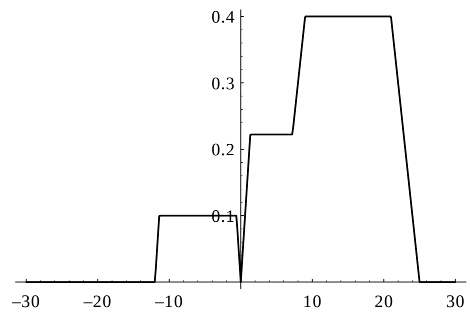
We can simplify this expression by removing the entries that are equivalent to zero:

```
In[53]:= fuzzyTheta[theta_] = % /. Min[0., _] -> 0

Out[53]= Max[{0, Min[0.1, NS[theta]], Min[0.1, PS[theta]],
           Min[0.222222, PS[theta]], Min[0.4, PM[theta]]}]
```

Here is the output fuzzy set:

```
In[54]:= Plot[fuzzyTheta[theta], {theta, -30, 30}]
```



Now, we must defuzzify this set to get a numerical value. To find the centroid of the output fuzzy set, we should integrate the product of θ and `fuzzyTheta` and divide by the integral of `fuzzyTheta`, both integrations taken over the range of θ. In the function `defuzzify`, I use an approximation that will speed the calculation while resulting in a small error:

```
In[55]:= defuzzify[fuzzySet_, {t_, min_, max_, dt_}]:=
           Sum[t fuzzySet, {t, min, max, dt}]/
             Sum[fuzzySet, {t, min, max, dt}]
```

The output θ value for our example is:

```
In[56]:= defuzzify[fuzzyTheta[theta], {theta, -30, 30, .5}]

Out[56]= 10.6783
```

The function `steer` combines the steps above to compute the steering angle θ for a given configuration (x, ϕ):

```
In[57]:= steer[x_, phi_]:=
         defuzzify[
           Max[Flatten[
             MapThread[Min,
               {Outer[Min, phiSets[phi], xSets[x]],
                 truckFAM[theta]}, 2] ]] /. Min[0., _] -> 0,
           {theta, -30, 30, .5} ]
```

The function `simulateTruck` takes the initial values of x, y, and ϕ, and computes a list of configurations $\{x, y, \phi\}$ giving a trajectory for the truck until it reaches a y value of at least 95:

```
In[58]:= simulateTruck[x0_, y0_, phi0_] :=
         Module[{x = x0, y = y0, phi = phi0, newPhi, result = {} },
           While[y <= 95.,
             newPhi = phi + steer[x, phi];
             AppendTo[result,
               {x, y, phi} =
                 {x + 5 Cos[newPhi Pi/180],
                  y + 5 Sin[newPhi Pi/180], newPhi} // N ]; ];
           result ]
```

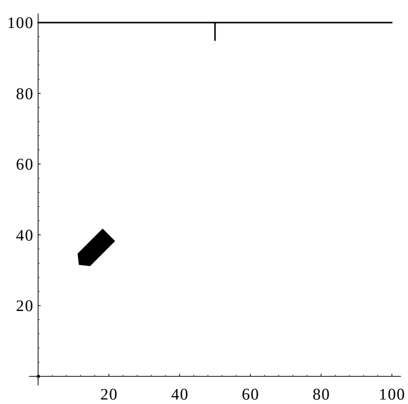
Note that the stepsize (5) used in this function can easily be changed.

The position of the truck can be displayed with the function showTruck. It takes as inputs the $\{x, y, \phi\}$ configuration and the length and width of the truck:

```
In[59]:= showTruck[{x_, y_, phi_}, {l_, w_}] :=
  Module[{s = Sin[phi Pi/180]//N, c = Cos[phi Pi/180]//N},
  Show[Graphics[{
    Polygon[Transpose[
      {x, y} + {{-s w/2, s w/2, s w/2 - c l,
          -1.2 c l, - s w/2 - c l, -s w/2},
        {c w/2, -c w/2, - c w/2 - s l,
          -1.2 s l, c w/2 - s l, c w/2}} ]],
    Point[{0,0}],
    Line[{{0, 100}, {100, 100}}],
    Line[{{50, 100}, {50, 95}}] },
    Axes -> True, AspectRatio -> Automatic,
    AxesOrigin -> {0, 0}]]];
```

For example:

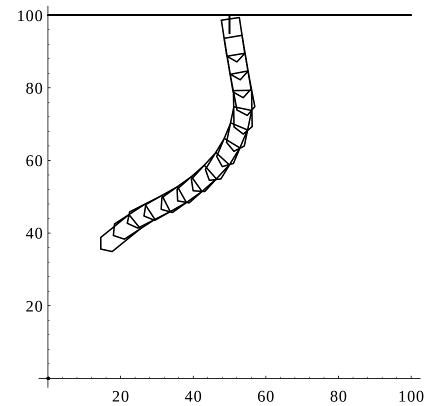
```
In[60]:= showTruck[{20, 40, 45}, {10, 5}]
```



Let's compute the trajectory of a truck starting at the position shown above:

```
In[61]:= simulateTruck[20, 40, 45]
Out[61]= {{23.8857, 43.1466, 39.}, {28.0791, 45.8698, 33.},
  {32.5341, 48.1397, 27.}, {36.8884, 50.5974, 29.441},
  {40.962, 53.4967, 35.441}, {44.7102, 56.806, 41.441},
  {48.0919, 60.4889, 47.441}, {50.698, 64.756, 58.5859},
  {52.6265, 69.3691, 67.3134}, {53.6452, 74.2642, 78.2441},
  {53.5617, 79.2635, 90.9564}, {52.6199, 84.174, 100.858},
  {51.7959, 89.1057, 99.4856}, {50.9721, 94.0373, 99.4829},
  {50.2237, 98.981, 98.6083}}
```

We can produce an animation of the moving truck by mapping the function showTruck[#, {10, 5}]& onto this list of configurations. The resulting graphics objects can also be displayed in one frame by using Show. The individual positions of the truck are easier to see if we replace Polygon by Line in the function showTruck:



I have not explored all of the various starting positions, so I cannot guarantee that every one will result in an acceptable trajectory. You can experiment by changing the various parameters, such as the boundaries of the fuzzy sets, the step-size in simulateTruck, and so on.

A fuzzy logic implementation of a control system is often a good choice when a mathematical model of the system is either unavailable or too complex to simulate efficiently. Our can example shows that it is fairly easy to construct a fuzzy controller for a system that might otherwise require quite an effort at mathematical modeling. Other interesting examples, both hypothetical and real-life, appear in [Terano et al. 1992, Kosko 1992, White and Sofge 1992].

References

Kosko, B. 1992. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence.* Prentice Hall, Englewood Cliffs, NJ.

Maeder, R. 1991. *Programming in Mathematica* 2d ed. Addison-Wesley, Reading, MA.

Terano, T., K. Asai, and M. Sugeno. 1992. *Fuzzy Systems Theory and Its Application.* Academic Press, Boston, MA.

White, David A., and Donald A. Sofge, eds. 1992. *Handbook of Intelligent Control.* Van Nostrand Reinhold, New York.

James A. Freeman
Loral Space Information Systems, Artificial Intelligence Laboratory, P.O. Box 58487, Mail Code F8H4A, Houston, TX 77258, (713) 335-6626
71340.3016@compuserve.com

The electronic supplement contains the notebook Fuzzy Control Systems.