ARMY RESEARCH LABORATORY

A Target-Tracking Algorithm

Michael J. Vrabel

ARL-TN-158 April 2000

Abstract

A computationally fast algorithm for tracking targets in a sequence of scenes is described. The algorithm is based on a variation of template matching.

Contents

Tables

1. Introduction

Among the plethora of automatic target recognition (ATR) techniques developed over the last several decades, one of the most basic is template matching. This report details a template-matching algorithm developed to track targets contained in a sequence of forward-looking infrared (FLIR) images. The two prime driving factors in algorithm development were tracking accuracy and computational burden.

2. Template-Matching Algorithm

The template-matching concept is elementary: A rectangular patch of image pixels containing a representation of the subject target is stepped across a scene, and the best match is assumed as the most probable location of the target.

2.1 Scene Rescaling and Preprocessing

One can rescale the pixels of a scene in a number of ways. Since images from the frame sequence for this study are the source of all templates and rescaling involves some computational penalty, no frame rescaling was tried.

Rather than just using a gray-scale image as the template basis, one can preprocess the templates and frames to enhance various image attributes and deemphasize others. Although any preprocessing scheme can increase computational requirements, its use can be justified where significant performance gains can be achieved. As a first try, a gradient model, based on the Sobel operator, was implemented. Initial results indicated that the gradient-based template performed so much worse than the gray-scale-based template that all further investigations into preprocessing were ended.

2.2 Background Correlation

The primary source of frame-to-frame target location variability for the test scene set is random frame dither. To estimate target location in any given frame with respect to background in the previous frame, one can use background correlation. For the test scene set, the maximum frame-to-frame variation of target location (based on ground-truth data) in either the horizontal or vertical directions is 10 pixels.

To compensate for frame dither, I used a rectangular template centered in any given frame, with a 15-pixel border. For the 128- $\times$ 128-pixel images, the corresponding template encompasses approximately 10,000 pixels—an exceedingly large template. To vary such a template over a conservatively chosen range of 15 pixels is, from a computational point of view, impractical. I found that such a template could be down sampled by selection of every fourth pixel vertically and horizontally (resulting in a background template of approximately 600 pixels) without adversely affecting performance accuracy. The template can also be shifted in steps of two pixels over the 15-pixel range—resulting in a further factor of four reduction in background correlation time. These reductions in computational requirements allow template matching to become a practical approach to aligning sequential frames.

2.3 Scene Search Region

With the given or computed location of a target in any frame, by using background correlation, one can estimate the target's location for the next frame. This would seem to permit a target search near the region of its estimated location. Such a localized search will reduce algorithm execution time and may also improve performance.

To search over increasingly smaller regions of a frame while minimizing the probability of the target falling outside the search region, I included a frame-to-frame variable search region capability into the algorithm. To implement this capability, I chose a nominal search region size for the data set. With the actual or computed target location in any frame (from the background correlator), the target's location in the next frame was estimated. If after the most probable location of the target on the second frame (within the given search region) was computed from the target template and if the estimated and calculated locations agreed, the search region size for the next frame continued unchanged. If this distance between the estimated and calculated locations grew, then the search region for the subsequent frame could be enlarged (to a maximum search area four times the nominal search area). If the calculated location of a target approached too closely to the border of the search region, the probability of error for that frame was assumed to have significantly increased. Because so many frames were in the sequence of images, not all were required to track a target; these high probability error frames were discarded, and the search continued to the next frame. I found that the cumulative effect of these procedures improved performance while reducing execution time.

2.4 Template Updating and Weighting

If you were to track any target through a sequence of frames, its appearance would progressively change. This change would have the obvious effect of degrading the performance of the template-matching algorithm. To compensate for this degradation, at least in part, the pixel values of the target template are progressively updated as the algorithm passes down the sequence of frames. Equation (1) describes how these values are updated:

$$T_{K+1}^{A}(n) = (\beta_1 T_K^A(n) + F_K^A(n))/(\beta_1 + 1) , \tag{1}$$

where

$$\begin{aligned} \beta_1 &= \text{user-defined constant,} \\ T(n) &= \text{the value of pixel } n \text{ of template, and} \\ F(n) &= \text{the value of pixel } n \text{ for the calculated location of target } A \\ & \text{ in frame } K. \end{aligned}$$

It is reasonable to assume that not all pixels of a template contribute equally to a template-matching algorithm's performance. For one to determine whether this unequal contribution is random from frame to frame or is (in some manner) systematic, template weighting was introduced. If this contribution is not highly random, then template weighting should improve performance. To implement the weighting, simply multiply the results of the template-frame match for each template pixel by the weight term. This weight is computed as (one should note the similarity between equations (1) and (2))

$$W_{K+1}^{A}(n) = (\beta_2 W_K^A(n) + a^A(n)a^A(n))/(\beta_2 + 1) , \tag{2}$$

where

$$\begin{aligned} a^A(n) &= (S - s^A(n))/S , \\ S &= \sum_n s^A(n) , \text{ and} \\ s^A(n) &= (T_K^A(n) - F_K^A(n))(T_K^A(n) - F_K^A(n)) . \end{aligned}$$

I selected this weight term after experimentation with several different forms of these equations.

3. Algorithm Performance

The performance of the template-matching algorithm was optimized and then tested with the test scene set. The L1816 test set has ground-truth values for all targets for frames 53 through 288. Parameter optimization and testing were done with this frame sequence. This sequence can be divided into two regions: (1) the beginning, where comparatively few pixels are on target and the frame-to-frame variation is minimal and (2) the end, where both the target pixel count and frame variations are much larger. I designed an optimum model for each region (model 1 and model 2). An optimum model is one that both maximizes performance and minimizes algorithm execution time. I also tried to keep as many parameters of each region's model the same as possible (see table 1).

The parameters are defined in terms of the variable names used in the code listed in the appendix. Where parameters were previously given alternate designations, these designations are given in parentheses (see table 1).

3.1 Model 1

This model is the parameter set designed to perform optimally for the frames at the beginning of the frame sequence. The performance of model 1 is demonstrated by defining a frame sequence, selecting the first frame as

Table 1. Algorithm parameters. Variable names are those of code listed in appendix (see function main).

Model 1						Model 2		
sample	=	1	sample	=	2	sample	=	2
bgnd	=	4	bgnd	=	4	bgnd	=	4
stepa	=	1	stepa	=	1	stepa	=	1
step	=	2	step	=	2	step	=	2
reset	=	1	reset	=	1	reset	=	1
edge	=	13	edge	=	20	edge	=	22
error	=	1	error	=	1	error	=	1
errora	=	1	errora	=	1	errora	=	1
mask	=	9	mask	=	11	mask	=	12.5
weight	=	1	weight	=	1	weight	=	1
wt (β_2)	=	6	wt (β_2)	=	6	wt (β_2)	=	6
time (β_1)	=	4	time (β_1)	=	4	time (β_1)	=	4
suppress	=	0	suppress	=	0	suppress	=	0

the source of all target templates, and testing its performance on the subsequent frames. Two targets are on each frame: the M60 and tnk. To increase the statistical significance of the results, I repeated model 1's demonstration of performance for a series of 40 frame sets. Each frame set was created by incrementing the previous frame sequence (and, hence, also the target template source) by one frame. For instance, if the first frame set encompassed frames 55 through 115 (with the template source as 55), then the second set would encompass 56 through 116 (with the template source as 56), and so on for 40 sets. The results for the 40 frame sets were then either averaged or summed.

To optimize the template-matching algorithm's parameters, I used a 20-frame set with each spanning 100 frames. The initial frame set encompassed frames 55 through 155. To test the performance of the optimized model, I used the aforementioned sequence of 40-frame sets. These sets spanned varying frame sequence lengths. Because both optimization and test data sets were drawn from the same database, I sought to introduce differences into the two sets. Besides the differences in frame set counts, varying frame sequence starting points (hence different target templates) were used. Nevertheless, one should recognize the overwhelming similarities between the optimization and test data sets and be very circumspect in interpreting the results. As more data sets are brought on-line, this problem should diminish.

Results for model 1 are presented for two different target templates. The first is for a template down sampled by selection of every fourth pixel (table 2) and the second is for no down sampling (table 3). The limited number of pixels on target for the data sets used with model 1 complicates down sampling and maintaining performance accuracy.

Table 2. Performance of down-sampled model 1. Code parameters listed in table 1.

			D^4		E^5		
A^1	B^2	C^3	M60	tnk	M60	tnk	both
55	120	33	16.7	26.5	27	13	12
55	100	25	12.9	16.1	28	16	15
55	80	17	8.9	8.9	29	23	21
55	60	9	4.2	4.6	29	27	24
75	40	5	7.8	2.8	25	33	25
95	20	1	2.3	0.7	30	37	30

[1]Beginning frame number of first frame sequence.
[2]Frame sequence length.
[3]Average number of skipped frames per sequence.
[4]Average number of bad target hits per sequence.
[5]Total number of error-free passes through sequence for 40-frame sequences.

Table 3. Performance of model 1 with no target down sampling. Code parameters listed in table 1.

			D^4		E^5		
A^1	B^2	C^3	M60	tnk	M60	tnk	both
55	120	3	2.6	29.8	38	2	2
55	100	2	2.6	12.9	38	11	11
55	80	1	1.9	3.3	38	28	28
55	60	1	0.9	0.9	38	38	38
75	40	3	0.6	4.4	37	32	32
95	20	1	0.1	1.5	39	34	34

[1]Beginning frame number of first frame sequence.
[2]Frame sequence length.
[3]Average number of skipped frames per sequence.
[4]Average number of bad target hits per sequence.
[5]Total number of error-free passes through sequence for 40-frame sequences.

Note that for models 1 and 2, the target template is significantly larger than the target. The inclusion of substantial background in the template while adversely affecting computer execution time significantly improved accuracy. Despite the large templates, where a match was achieved, the difference between estimated and ground-truth location of targets on average was typically one or two pixels and with the worst-case fit being rarely more than six pixels.

3.2 Model 2

Model 2 was optimized for the frames near the end of the data set sequence. Its performance is given in table 4. I used 20 frame sets to optimize the parameters of this model. The frame sequence for the first of these frame sets was 185 through 245. Template updating has a significant effect on model performance. This is shown in table 5.

The test configuration was that of row 4 of table 4 with parameter time (β_1) the variable. Note that template weighting minimally affects performance and can probably be deleted from the model.

Table 4. Performance of down-sampled model 2. Code parameters listed in table 1.

A^1	B^2	C^3	D^4 M60	D^4 tnk	E^5 M60	E^5 tnk	E^5 both
65	180	—	20.0	10.6	7	8	7
85	160	79	10.9	6.6	27	19	18
105	140	28	1.8	3.7	38	27	27
125	120	22	0.0	0.8	39	37	37
145	100	22	0.0	0.0	40	40	40
165	80	22	0.0	0.0	40	40	40
185	60	22	0.0	0.0	40	40	40

[1]Beginning frame number of first frame sequence.
[2]Frame sequence length.
[3]Average number of skipped frames per sequence.
[4]Average number of bad target hits per sequence.
[5]Total number of error-free passes through sequence for 40-frame sequences.

Table 5. Effect of template-updating parameters (β_1) on performance of model 2.

Time[1]	C^2	D^3 M60	D^3 tnk	E^4 M60	E^4 tnk	E^4 both
10,000	78	16.4	8.8	12	17	12
20	68	19.7	13.4	11	14	10
10	29	2.2	15.9	32	9	9
8	21	3.5	12.2	27	16	16
6	20	2.1	3.8	31	30	30
4	22	0.0	0.8	39	37	37
3	22	0.3	0.7	37	36	36

[1]See appendix (or β_1 of eq (1)) for definition of time.
[2]Average number of skipped frames per sequence.
[3]Average number of bad target hits per sequence.
[4]Total number of error-free passes through sequence for 40-frame sequences.

4. Discussion

It is feasible to develop both a reasonably accurate and a computationally efficient algorithm for target tracking based on template matching. The frame-to-frame redundancy in a sequence of images is an attribute that is exploited by the present version of template matching to maximize tracking accuracy. As has been demonstrated, much that is potentially computationally inefficient about a template-matching scheme can be circumvented through careful algorithm design. As is too often the case in the target recognition field, a definitive statement cannot be made about absolute algorithm accuracy. Defining an absolute performance standard requires not only a more extensive set of image sequences than was available for this study but also several alternative tracking algorithms for comparison.

Appendix. Target-Tracking Algorithm Source Code

The following is a C source code listing of the template-matching algorithm of this report.

```
/* template.c     template matching code for the SDF data sets  */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#define size 300
#define fsize 128
#define tcount 2

int seq, start, stop, gtx[tcount][size], gty[tcount][size], alt, counter, dot;
int sample, row, col, rate, edge, tplate[tcount][50][50], error, errora, ed;
int x[tcount][size], y[tcount][size], tgtx[size], tgty[size], sing, dcount;
int xshift, yshift, weight, time, bgnd, step, reset, bad[tcount], stepa, skip;
int wrong[tcount], recover[tcount], suppress, wt, box, skip_seq[200];
int inc, pass, sta, sto, kount[tcount],efree[tcount][500];
float mse[tcount][fsize][fsize], back[tcount], target[tcount], mask, dist;
float frame1[2][fsize][fsize], mask1[tcount][size], w[tcount][50][50], bigd;
float bad_d[tcount];
char isource[7], image[12];
double frame[fsize][fsize];
unsigned char im[fsize][fsize];


read_ground_truth()
{
int na, nb, nc, ka, xx, yy, big;
char stringa[12], stringb[5], stringc[12], temp[81];
FILE *in_file1, *in_file2;

   for(nb=0; nb<tcount; nb++){
   for(na=0; na<size; na++){
```

```
gtx[nb][na]=-500;
gty[nb][na]=-500;
}}

if(seq==1){
  in_file1=fopen("/net/ragu/export/data/sdf/motion/L1816/L1816m60.gt2","r");
  in_file2=fopen("/net/ragu/export/data/sdf/motion/L1816/L1816tnk.gt2","r");
/* presently have gt values in above files for frames 053 thru 288  */
  for(na=0; na<1000; na++){
  for(nc=0; nc<2; nc++){
  if(nc==0){
    fscanf(in_file1,"%s %s %d %d",&stringa, &stringb, &xx, &yy);
    if(feof(in_file1))goto jumpa;
/* following line stores enough characters to later find image source */
    if(na==0)for(nb=0; nb<7; nb++)isource[nb]=stringa[nb];
    for(nb=0; nb<5; nb++)stringb[nb]=stringa[nb+7];
    ka=atoi(stringb);
/* xx and yy contain x and y ground truth locations... note corrections to
   the values stored in the ground truth tables  */
    gtx[nc][ka]=xx;
    gty[nc][ka]=fsize-yy;
    fgets(temp, 80, in_file1);
  }
  if(nc==1){
    fscanf(in_file2,"%s %s %d %d",&stringc, &stringb, &xx, &yy);
    if(strcmp(stringa,stringc)!=0){
      printf("PROBLEM WITH GROUND TRUTH DATA\n");
      exit(1);
    }
    gtx[nc][ka]=xx;
    gty[nc][ka]=fsize-yy;
    fgets(temp, 80, in_file2);
  }
  }
  }
  jumpa:
  fclose(in_file1);
```

```
     fclose(in_file2);

/* find maximum frame-to-frame axis shift for gt values:  */
/*******************************************
   big=0;
   for(na=0; na<tcount; na++){
   for(nb=53; nb<288; nb++){
   ka=nb+1;
   if(abs(gtx[na][nb]-gtx[na][ka])>big)big=abs(gtx[na][nb]-gtx[na][ka]);
   if(abs(gty[na][nb]-gty[na][ka])>big)big=abs(gty[na][nb]-gty[na][ka]);
   }}
   printf("largest frame-to-frame gt axis shift= %d\n",big);
********************************************/
   }
}


get_frame(na)
int na;
{
int nb, nc, ka=0, kb=0;
double big, small;
char temp[81], string[100], stringa[4] ;
FILE *in_file, *out_file;

   if(seq==1){
      if(na<10)sprintf(stringa,"000%d",na);
      if(na>9 && na<100)sprintf(stringa,"00%d",na);
      if(na>99 && na<1000)sprintf(stringa,"0%d",na);
      if(na>999)sprintf(stringa,"%d",na);
      strcpy(image,isource);
      strcat(image,stringa);
 /* image[] contains frame na file name  */
    }
    sprintf(string,"/net/ragu/export/data/sdf/motion/Locaas/%s_r1.bin",image);
    in_file=fopen(string,"rb");
    if(na==start){
       big=-1.E10;
       small=1.E10;
```

```
      }
    for(nb=0; nb<3; nb++)fgets(temp, 80, in_file);
    for(nb=0; nb<fsize; nb++){
    for(nc=0; nc<fsize; nc++){
    fread(&im[nb][nc],sizeof(unsigned char),1,in_file);
    frame[nb][nc]=(double)im[nb][nc];
    frame1[0][nb][nc]=frame1[1][nb][nc];
    frame1[1][nb][nc]=frame[nb][nc];
    if(na==start){
      if(frame[nb][nc]<small)small=frame[nb][nc];
      if(frame[nb][nc]>big)big=frame[nb][nc];
      }
    }}
    fclose(in_file);
/* can use following block to generate a single MATLAB image  */
/*
    if(na==start){
      out_file=fopen("scene.dat","w");
      for(nb=0; nb<fsize; nb++){
      for(nc=0; nc<fsize; nc++){
      fprintf(out_file,"%d ",(int)(100.*(big-frame[nb][nc])/(big-small)));
      }
      fprintf(out_file,"\n");
      }
      fclose(out_file);
    }
*/
}


down_sample_frame(ka)
int ka;
{
int na, nb, ma, mb;

    ma=-1;
    for(na=0; na<fsize; na++){
    if(na%sample==0){
      ma+=1;
```

```
    mb=-1;
  }
  for(nb=0; nb<fsize; nb++){
  if(na%sample==0 && nb%sample==0){
    mb+=1;
    frame[ma][mb]=frame[na][nb];
  }
  }}
/* row and col contain the row and column count for the down sampled frame: */
}


alternate_down_sample_frame(ka)
int ka;
{
int na, nb, ma, mb;

  ma=-1;
  for(na=0; na<fsize; na++){
  if(na%sample==0){
    ma+=1;
    mb=-1;
  }
  for(nb=0; nb<fsize; nb++){
  if(na%sample==0 && nb%sample==0){
    mb+=1;
    if(na>0 && na<fsize-1 && nb>0 && nb<fsize-1)
      frame[ma][mb]=0.4*frame[na][nb]+0.1*frame[na-1][nb]+0.1*frame[na+1][nb]+
      0.1*frame[na][nb-1]+0.1*frame[na][nb+1]+0.05*frame[na-1][nb-1]+
      0.05*frame[na-1][nb+1]+0.05*frame[na+1][nb-1]+0.05*frame[na+1][nb+1];
    else
      frame[ma][mb]=frame[na][nb];
  }
  }}
/* row and col contain the row and column count for the down sampled frame: */
  if(ka==start+1){
    row=ma+1;
    col=mb+1;
  }
}
```

```
templates_from_gt(ka)
int ka;
{
   int na, nb, nc, ma, mb, ia, ib;

   for(nc=0; nc<tcount; nc++){
/* following block allows the first frame thru template matching to be masked */
   if(ka==start || reset==1){
      x[nc][ka]=gtx[nc][ka]/sample;
      y[nc][ka]=gty[nc][ka]/sample;
   }
   ma=gty[nc][ka]/sample-edge/2;
   if(ma<0)ma=0;
   if((ma+edge)>col)ma=col-edge;
   mb=gtx[nc][ka]/sample-edge/2;
   if(mb<0)mb=0;
   if((mb+edge)>row)mb=row-edge;
   ia=-1;
   for(na=ma; na<ma+edge; na++){
   ib=-1;
   ia+=1;
   for(nb=mb; nb<mb+edge; nb++){
   ib+=1;
/*   tplate[nc][ia][ib]=(int)(0.5*(frame1[0][na][nb]+frame1[1][na][nb])); */
   tplate[nc][ia][ib]=frame[na][nb];
   }}
   }
}



do_template_matching(nc,naa)
int nc, naa;
{
   int na, nb, nd, ka, kb, ma, mb, ja, jb, jc, jd, xx, yy;
   float aa, m, temp, low, ax, ay, a;
```

```
low=1.E+40;
m=2.0*mask*mask1[nc][naa];
for(na=0; na<row; na++){
for(nb=0; nb<col; nb++){
mse[nc][na][nb]=-1.0;
}}
ja=(int)(y[nc][naa-1]+yshift/sample-row/m);
jb=(int)(x[nc][naa-1]+xshift/sample-col/m);
jc=(int)(y[nc][naa-1]+yshift/sample+row/m);
jd=(int)(x[nc][naa-1]+xshift/sample+col/m);


for(na=ja; na<=jc; na+=stepa){
if(na<edge/2+1 || na>=row-1-edge/2)goto jumpa;
for(nb=jb; nb<=jd; nb+=stepa){
if(nb<edge/2+1 || nb>=col-1-edge/2)goto jumpb;
temp=0.0;
aa=0.0;
ma=-1;
for(ka=na-edge/2; ka<na-edge/2+edge; ka++){
ma+=1;
mb=-1;
for(kb=nb-edge/2; kb<nb-edge/2+edge; kb++){
mb+=1;
    if(error==0){
      temp+=w[nc][ma][mb]*fabs(tplate[nc][ma][mb]-frame[ka][kb]);
      aa+=1;
      }
   if(error==1){
      temp+=w[nc][ma][mb]*(tplate[nc][ma][mb]-frame[ka][kb])*
      (tplate[nc][ma][mb]-frame[ka][kb]);
      aa+=1;
      }
 }
 }
 mse[nc][na][nb]=temp/aa;
 if(mse[nc][na][nb]<low){
    low=mse[nc][na][nb];
```

```
      xx=nb;
      yy=na;
   }
   jumpb:
   }
   jumpa:
   }

/* make next search box a function of location of target in present box: */
   ax=fabs((x[nc][naa-1]+xshift/sample-xx)/(col/(mask*mask1[nc][naa])));
   ay=fabs((y[nc][naa-1]+yshift/sample-yy)/(row/(mask*mask1[nc][naa])));
   if(ax>ay)
      a=ax;
   else
      a=ay;
/*   printf("counter: %d  a[%d]= %.3f  ",counter,nc,a[nc]);   */
   if(a>0.45 && suppress==0){
      skip=1;
/*      printf("SKIPPING FRAME # %d\n",naa);  */
      goto jump;
      }
/*   printf("\n");  */
   mask1[nc][naa+1]=(1.0-a);
   x[nc][naa]=xx;
   y[nc][naa]=yy;
   jump:
}


predict_target_location()
{
   int na, nb, ma, mb, ka, kb, kc;
   float temp, tempa;

/* use background correlation to find frame to frame dither
   ka is maximum anticipated axis shift
   kb is template matching step size
   kc relates to pixel count in background template */
   temp=1.E+40;
```

```
 ka=15;
 kb=step;
 kc=bgnd;
 for(na=-ka; na<=ka; na+=kb){
 for(nb=-ka; nb<=ka; nb+=kb){
 tempa=0.0;
 for(ma=ka; ma<fsize-ka; ma+=kc){
 for(mb=ka; mb<fsize-ka; mb+=kc){
 if(errora==0)
    tempa+=fabs(frame1[1][ma][mb]-frame1[0][ma+na][mb+nb]);
 if(errora==1)
    tempa+=(frame1[1][ma][mb]-frame1[0][ma+na][mb+nb])*
           (frame1[1][ma][mb]-frame1[0][ma+na][mb+nb]);
 }}
 if(tempa<temp){
    temp=tempa;
    xshift=nb;
    yshift=na;
 }
 }}
}


detected_target_weighting(naa)
int naa;
{
int na, nb, nc, ma, mb;
float temp[tcount][50][50], tempa[tcount], a;

  for(nc=0; nc<tcount; nc++){
  tempa[nc]=0.0;
  ma=-1;
  for(na=y[nc][naa]-edge/2; na<y[nc][naa]-edge/2+edge; na++){
  ma+=1;
  mb=-1;
  for(nb=x[nc][naa]-edge/2; nb<x[nc][naa]-edge/2+edge; nb++){
  mb+=1;
  temp[nc][ma][mb]=1.0;
  if(na>=0 && na<row && nb>=0 && nb<col)
```

```
        temp[nc][ma][mb]=(tplate[nc][ma][mb]-frame[na][nb])*
        (tplate[nc][ma][mb]-frame[na][nb]);
    tempa[nc]+=temp[nc][ma][mb];
    }}
    }
    for(nc=0; nc<tcount; nc++){
    for(na=0; na<edge; na++){
    for(nb=0; nb<edge; nb++){
    a=(tempa[nc]-temp[nc][na][nb])/tempa[nc];
    w[nc][na][nb]=(wt*w[nc][na][nb]+a*a)/(wt+1.0);
    }}}
/* adjust below term to appropriate value to print weight set */
    if(naa%4000==0){
        for(nc=0; nc<tcount; nc++){
        for(na=0; na<edge; na++){
        for(nb=0; nb<edge; nb++){
        printf("%.3f ",w[nc][na][nb]);
        }
        printf("\n");
        }
        printf("\n\n");
        }
    }
}


generate_averaged_template(naa)
int naa;
{
int na, nb, nc, ma, mb, t;

    t=time;
/*   if(counter%rate==1)t=1;  */
    for(nc=0; nc<tcount; nc++){
    ma=-1;
    for(na=y[nc][naa]-edge/2; na<y[nc][naa]-edge/2+edge; na++){
    ma+=1;
    mb=-1;
    for(nb=x[nc][naa]-edge/2; nb<x[nc][naa]-edge/2+edge; nb++){
```

```
  mb+=1;
  if(na>=0 && na<row && nb>=0 && nb<col)
     tplate[nc][ma][mb]=(t*tplate[nc][ma][mb]+frame[na][nb])/(t+1.0);
  }}
  }
}



generate_performance_statistics(ka,kb)
int ka, kb;
{
int nc;
float temp, er;

   dot=0;
   for(nc=0; nc<tcount; nc++){
   if(counter==1)wrong[nc]=0;
   er=sqrt(
   (1.0*gtx[nc][ka]/sample-x[nc][ka])*(1.0*gtx[nc][ka]/sample-x[nc][ka])+
   (1.0*gty[nc][ka]/sample-y[nc][ka])*(1.0*gty[nc][ka]/sample-y[nc][ka]));
   if(er>0.5*edge){
        efree[nc][kb]=1;
        bad[nc]+=1;
        bad_d[nc]+=er;
        dot=1;
        if(wrong[nc]==0)kount[nc]+=1;
        wrong[nc]=1;
/*      printf("bad hit on %d at counter= %d error/edge= %.3f edge= %d\n"
    ,nc,counter,er/edge,edge);   */
    }
   else{
        if((counter-1)%rate!=0 && wrong[nc]==1){
            recover[nc]+=1;
            wrong[nc]=0;
            }
        wrong[nc]=0;
        temp=sqrt((1.0*gtx[nc][ka]/sample-x[nc][ka])*
        (1.0*gtx[nc][ka]/sample-x[nc][ka])+
```

```
        (1.0*gty[nc][ka]/sample-y[nc][ka])*(1.0*gty[nc][ka]/sample-y[nc][ka]));
        dist+=temp;
        if(temp>bigd)bigd=temp;
        dcount+=1;
        }
  }
}


make_MATLAB_movie(naa)
int naa;
{

int na, nb, nc, ma, ka, kb, temp[fsize][fsize];
float big, small, m;
char string[100];
FILE *out_file;

   big=-1.E10;
   small=1.E10;
   for(na=0; na<row; na++){
   for(nb=0; nb<col; nb++){
   if(frame[na][nb]>big)big=frame[na][nb];
   if(frame[na][nb]<small)small=frame[na][nb];
   }}
   for(na=0; na<row; na++){
   for(nb=0; nb<col; nb++){
   temp[na][nb]=100.*((frame[na][nb]-small)/(big-small));
   }}

/* box gt location of targets:  */
   for(nc=0; nc<tcount; nc++){
   m=2.*mask*mask1[nc][naa];
   ma=100;
   if(nc==1)ma=20;
   for(na=gtx[nc][naa]/sample-edge/2-1;
       na<gtx[nc][naa]/sample-edge/2+edge+1; na++){
       temp[gty[nc][naa]/sample+edge/2+1][na]=ma;
       temp[gty[nc][naa]/sample+edge/2-edge-1][na]=ma;
```

```
  }
  for(na=gty[nc][naa]/sample-edge/2-1;
      na<gty[nc][naa]/sample-edge/2+edge+1; na++){
      temp[na][gtx[nc][naa]/sample+edge/2+1]=ma;
      temp[na][gtx[nc][naa]/sample+edge/2-edge-1]=ma;
  }
/* cross hairs on best template matching fit:  */
  for(nb=0; nb<6; nb++){
     if((y[nc][naa]-edge/2-nb)>=0)temp[y[nc][naa]-edge/2-nb][x[nc][naa]]=ma;
     if((y[nc][naa]+edge/2+nb)<row)temp[y[nc][naa]+edge/2+nb][x[nc][naa]]=ma;
     if((x[nc][naa]-edge/2-nb)>=0)temp[y[nc][naa]][x[nc][naa]-edge/2-nb]=ma;
     if((x[nc][naa]+edge/2+nb)<col)temp[y[nc][naa]][x[nc][naa]+edge/2+nb]=ma;
  }
/* dotted line around search box for each target based on value of mask */
  if(mask>=0.01 && box==0){
  for(na=x[nc][naa-1]+xshift/sample-col/m;
      na<x[nc][naa-1]+xshift/sample+col/m; na+=2){
      if(na>=0 && na<col){
         if((y[nc][naa-1]+yshift/sample+row/m)<row)
            temp[(int)(y[nc][naa-1]+yshift/sample+row/m)][na]=ma;
         if((y[nc][naa-1]+yshift/sample-row/m)>=0)
            temp[(int)(y[nc][naa-1]+yshift/sample-row/m)][na]=ma;
      }
  }
  for(na=y[nc][naa-1]+yshift/sample-row/m;
      na<y[nc][naa-1]+yshift/sample+row/m; na+=2){
      if(na>=0 && na<row){
         if((x[nc][naa-1]+xshift/sample+col/m)<col)
            temp[na][(int)(x[nc][naa-1]+xshift/sample+col/m)]=ma;
         if((x[nc][naa-1]+xshift/sample-col/m)>=0)
            temp[na][(int)(x[nc][naa-1]+xshift/sample-col/m)]=ma;
      }
  }
  }
/* if bad hit on frame, place large dot in corner */
  if(dot==1){
    for(ka=0; ka<8; ka++){
```

```
    for(kb=0; kb<8; kb++){
    temp[ka][kb]=ma;
    }}
    }
  }

  sprintf(string,"scene.dat%d",counter);
  out_file=fopen(string,"w");
  for(na=0; na<row; na++){
  for(nb=0; nb<col; nb++){
  fprintf(out_file,"%d ",temp[na][nb]);
  }
  fprintf(out_file,"\n");
  }
  fclose(out_file);
}


get_single_MATLAB_image()
{
/* creates side-by-side down sampled frame and template matching response
  with targets (from gt) boxed  */

int na, nb, nc, nd, temp[tcount+1][fsize][fsize];
float big, small;
FILE *out_file;

  for(nc=0; nc<tcount; nc++){
  big=-1.E10;
  small=1.E10;
  for(na=0; na<row; na++){
  for(nb=0; nb<col; nb++){
  if(mse[nc][na][nb]>big)big=mse[nc][na][nb];
  if(mse[nc][na][nb]<small && mse[nc][na][nb]>=0.0)small=mse[nc][na][nb];
  }}
  for(na=0; na<row; na++){
  for(nb=0; nb<col; nb++){
  if(mse[nc][na][nb]>=0.0)
     temp[nc][na][nb]=100.*((mse[nc][na][nb]-small)/(big-small));
```

```
else
  temp[nc][na][nb]=50.;
}}
}

big=-1.E10;
small=1.E10;
for(na=0; na<row; na++){
for(nb=0; nb<col; nb++){
if(frame[na][nb]>big)big=frame[na][nb];
if(frame[na][nb]<small)small=frame[na][nb];
}}
for(na=0; na<row; na++){
for(nb=0; nb<col; nb++){
temp[tcount][na][nb]=100.*((frame[na][nb]-small)/(big-small));
}}

for(nd=0; nd<tcount+1; nd++){
for(nc=0; nc<tcount; nc++){
for(na=gtx[nc][sing]/sample-edge/2-1;
    na<gtx[nc][sing]/sample-edge/2+edge+1; na++){
    temp[nd][gty[nc][sing]/sample+edge/2+1][na]=100;
    temp[nd][gty[nc][sing]/sample+edge/2-edge-1][na]=100;
}
for(na=gty[nc][sing]/sample-edge/2-1;
    na<gty[nc][sing]/sample-edge/2+edge+1; na++){
    temp[nd][na][gtx[nc][sing]/sample+edge/2+1]=100;
    temp[nd][na][gtx[nc][sing]/sample+edge/2-edge-1]=100;
}
}
}

out_file=fopen("scene.dat","w");
for(na=0; na<row; na++){
for(nc=0; nc<tcount+1; nc++){
for(nb=0; nb<col; nb++){
fprintf(out_file,"%d ",temp[nc][na][nb]);
}
}
```

```
   fprintf(out_file,"\n");
   }
}




main()
{
int na, nb, nc, nd, ka, kb[tcount], kc, ja, jb, ma, quit, total;
FILE *out_file;

   printf("image data set sequence? 1=1816\n");
   scanf("%d",&seq);
   printf("start and end sequence of frames at frames number:\n");
   scanf("%d %d",&start,&stop);
   sta=start;
   sto=stop;
   printf("increment size between passes thru data set?\n");
   scanf("%d",&inc);
   pass=1;
   if(inc>0){
      printf("number of passes?\n");
      scanf("%d",&pass);
      }
   printf("down sample frames by selecting:\n");
   printf("every pixel (1) every second (2) every third (3)...?\n");
   scanf("%d",&sample);
   if(sample>1){
      printf("select down sample mode: A= 1   B= 2\n");
      scanf("%d",&alt);
   }
   printf("corresponding sample rate for background template:\n");
   scanf("%d",&bgnd);
   printf("foreground pixel stepping rate:\n");
   scanf("%d",&stepa);
   printf("background pixel stepping rate:\n");
   scanf("%d",&step);
```

```
printf("reset target to gt value for each new template? no=0  yes=1\n");
scanf("%d",&reset);
printf("square templates: edge length in pixels before down sampling?\n");
scanf("%d",&edge);
ed=edge;
edge/=sample;
printf("in down sampled pixels: \n");
printf("form of template match? error=0  squared error=1 \n");
scanf("%d",&error);
printf("and for background: error=0  squared error=1\n");
scanf("%d",&errora);
printf("mask frame away from predicted target location\n");
printf("no= 0.5  yes= divide axis lengths by:\n");
scanf("%f",&mask);
printf("include template pixel weighting?  no=0  yes=1\n");
scanf("%d",&weight);
if(weight==1){
   printf("weighting time constant (int)?\n");
   scanf("%d",&wt);
   }
printf("time constant for template averaging (int):\n");
scanf("%d",&time);
printf("1st frame always selected in the sequence as a template source\n");
printf("template selection rate? (e.g. 5= every fifth frame)\n");
scanf("%d",&rate);
printf("suppress frame skipping? no=0  yes=1\n");
scanf("%d",&suppress);
printf("plot MATLAB single frame with corresponding template response\n");
printf("and gt locations? no= 0  yes= frame number  or, alternatively,\n");
printf("make (down sampled) frames for MATLAB movie: input -1\n");
scanf("%d",&sing);
if(sing==-1){
  printf("remove search box from movie frames? no=0  yes=1\n");
  scanf("%d",&box);
  }

read_ground_truth();
```

```
for(nc=0; nc<200; nc++)skip_seq[nc]=0;
jb=0;
for(nc=0; nc<tcount; nc++){
bad[nc]=0;
bad_d[nc]=0;
wrong[nc]=0;
recover[nc]=0;
}
total=0;
dist=0.0;
dcount=0;
bigd=0.0;
for(na=0; na<tcount; na++){
for(nb=start; nb<stop+1; nb++){
if(gtx[na][nb]<0 || gty[na][nb]<0){
   printf("WARNING...SOME FRAMES IN SEQUENCE MISSING GT DATA\n");
   printf("EXIT? YES=1  NO=0\n");
   scanf("%d",&quit);
   if(quit==1)exit(1);
}
}}

for(ma=0; ma<tcount; ma++)kount[ma]=0;
for(na=0; na<tcount; na++){
for(ma=0; ma<pass; ma++){
efree[na][ma]=0;
}}
get_frame(start-1);
start-=inc;
stop-=inc;
for(ma=0; ma<pass; ma++){
printf(" pass count= %d\n",ma);
start+=inc;
stop+=inc;
if(seq==1 && stop>288)stop=288;
for(na=0; na<tcount; na++){
for(nb=0; nb<size; nb++){
mask1[na][nb]=1.0;
```

```
}}
for(nc=0; nc<tcount; nc++){
for(na=0; na<50; na++){
for(nb=0; nb<50; nb++){
w[nc][na][nb]=1.0;
}}}
if(sample==1){
   row=fsize;
   col=fsize;
}
if(sample>1){
   row=fsize/sample+1;
   col=row;
 }
get_frame(start);
if(sample>1)down_sample_frame(start);
templates_from_gt(start);
counter=0;
na=start;
skip=0;
jb=0;
for(ja=start+1; ja<=stop; ja++){
 jumpa:
 if(skip==1)jb+=1;
 na+=1;
 if(na>=stop){
    skip_seq[jb]+=1;
    goto jumpb;
 }
 get_frame(na);
 predict_target_location();
 if(sample>1 && alt==1)down_sample_frame(na);
 if(sample>1 && alt==2)alternate_down_sample_frame(na);
 skip=0;
 for(nd=0; nd<tcount; nd++){
 do_template_matching(nd,na);
 if(skip==1){
    for(nb=0; nb<tcount; nb++){
```

```
        mask1[nb][na+1]=0.5;
        x[nb][na]=x[nb][na-1];
        y[nb][na]=y[nb][na-1];
        }
        for(nb=0; nb<fsize; nb++){
        for(nc=0; nc<fsize; nc++){
        frame1[1][nb][nc]=frame1[0][nb][nc];
        }}
        goto jumpa;
      }
      }
      total+=1;
      counter+=1;
      skip_seq[jb]+=1;
      jb=0;
      if(weight==1)detected_target_weighting(na);
      generate_averaged_template(na);
      generate_performance_statistics(na,ma);
      if(sing>0 && na==sing)get_single_MATLAB_image();
      if(sing==-1)make_MATLAB_movie(na);
      if(counter%rate==0)templates_from_gt(na);
      if(na>=stop)goto jumpb;
     }
      jumpb:
  }

      for(na=0; na<tcount; na++)kb[na]=0;
      for(na=0; na<tcount; na++){
      for(ma=0; ma<pass; ma++){
      if(efree[na][ma]==0)kb[na]+=1;
      }}
      ka=0;
      for(ma=0; ma<pass; ma++){
      kc=0;
      for(na=0; na<tcount; na++){
      if(efree[na][ma]!=0)kc=1;
      }
      if(kc==0)ka+=1;
```

```
}

if(seq==1 && stop==288)
   printf("WARNING...may have exceeded limit with parameter: stop\n");
printf("following results are average values per pass thru sequence\n");
printf("number of error free passes thru sequence= %d\n",ka);
printf("total frame count per pass= %.1f\n",(float)total/pass);
for(nc=0; nc<tcount; nc++){
printf("target %d  number of error free passes thru seq.= %d\n",nc,kb[nc]);
printf("target %d  bad hit count= %.1f\n",nc,(float)bad[nc]/pass);
printf("target %d  avg bad hit pixel distance from gt= %.1f\n",nc,
       bad_d[nc]/(bad[nc]+0.001));
printf("target %d  recovered bad sequence count= %.2f\n",nc,
(float)recover[nc]/pass);
printf("target %d  bad sequence count= %.2f\n",nc,
(float)kount[nc]/pass);
}
printf("avg. detected target gt distance in down sampled frame= %.2f\n",
dist/dcount);
printf("largest detected target - gt distance= %.2f\n",bigd);
printf("total target hit count per pass= %d\n",dcount/pass);
printf("consecutive frame skip sequence length vs total population\n");
for(nc=0; nc<200; nc++)if(skip_seq[nc]>0)printf("%d=%d  ",nc,skip_seq[nc]);
printf("\n\n");

out_file=fopen("template.dat","w");
fprintf(out_file,"test performance of template.c (17Sept98 version)\n");
fprintf(out_file," pass %d\n inc %d\n",pass,inc);
fprintf(out_file," seq %d\n start(initial) %d\n stop %d\n sample %d\n bgnd %d
step %d\n reset %d\n edge(original) %d\n error %d\n weight %d\n time %d
mask %.3f\n",seq,sta,sto,sample,bgnd,step,reset,ed,error,weight,time,mask);
 if(sample>1)fprintf(out_file," alt %d\n",alt);
 fprintf(out_file," errora %d\n rate %d\n",errora,rate);
 fprintf(out_file," stepa %d\n suppress %d\n",stepa,suppress);
 if(weight==1)
 fprintf(out_file," wt %d\n",wt);
 fprintf(out_file,"\n total frame count per pass= %d\n",total/pass);
```

```
fprintf(out_file,"following data is average value per pass thru sequence\n");
fprintf(out_file,"number of error free passes thru sequence= %d\n",ka);
for(nc=0; nc<tcount; nc++){
fprintf(out_file," target %d  number of error free passes thru seq.= %d\n",
nc,kb[nc]);
fprintf(out_file," target %d  bad hit count= %.1f\n",nc,
(float)bad[nc]/pass);
fprintf(out_file," target %d  recovered bad sequence count= %.2f\n",nc,
(float)recover[nc]/pass);
fprintf(out_file," target %d  bad sequence count= %.2f\n",nc,
(float)kount[nc]/pass);
}
fprintf(out_file,
"avg. detected target gt distance in down sampled frame= %.2f\n",dist/dcount);
fprintf(out_file,"largest detected target - gt distance= %.2f\n",bigd);
fclose(out_file);
}
```

Distribution

Admnstr
Defns Techl Info Ctr
Attn DTIC-OCP
8725 John J Kingman Rd Ste 0944
FT Belvoir VA 22060-6218

Ofc of the Secy of Defns
Attn ODDRE (R&AT)
The Pentagon
Washington DC 20301-3080

Ofc of the Secy of Defns
Attn OUSD(A&T)/ODDR&E(R) R J Trew
3080 Defense Pentagon
Washington DC 20301-7100

AMCOM MRDEC
Attn AMSMI-RD W C McCorkle
Attn AMSMI-RD-MG-IP R Sims
Redstone Arsenal AL 35898-5240

CECOM
Attn PM GPS COL S Young
FT Monmouth NJ 07703

Dir for MANPRINT
Ofc of the Deputy Chief of Staff for Prsnnl
Attn J Hiller
The Pentagon Rm 2C733
Washington DC 20301-0300

Night Vsn & Elect Sensors Dir
Attn AMSEL-RD-NVOD L Garn
10221 Burbeck Rd Ste 430
FT Belvoir VA 22060-5806

US Army ARDEC
Attn AMSTA-AR-TD M Fisette
Bldg 1
Picatinny Arsenal NJ 07806-5000

US Army Info Sys Engrg Cmnd
Attn ASQB-OTD F Jenia
FT Huachuca AZ 85613-5300

US Army Natick RDEC
Acting Techl Dir
Attn SSCNC-T P Brandler
Natick MA 01760-5002

US Army Simulation, Train, & Instrmntn Cmnd
Attn J Stahl
12350 Research Parkway
Orlando FL 32826-3726

US Army Tank-Automtv Cmnd Rsrch, Dev, & Engrg Ctr
Attn AMSTA-TR J Chapin
Warren MI 48397-5000

US Army Train & Doctrine Cmnd
Battle Lab Integration & Techl Dirctrt
Attn ATCD-B J A Klevecz
FT Monroe VA 23651-5850

US Military Academy
Mathematical Sci Ctr of Excellence
Attn MDN-A LTC M D Phillips
Dept of Mathematical Sci Thayer Hall
West Point NY 10996-1786

Nav Surface Warfare Ctr
Attn Code B07 J Pennella
17320 Dahlgren Rd Bldg 1470 Rm 1101
Dahlgren VA 22448-5100

DARPA
Attn S Welby
3701 N Fairfax Dr
Arlington VA 22203-1714

Hicks & Associates Inc
Attn G Singley III
1710 Goodrich Dr Ste 1300
McLean VA 22102

Palisades Inst for Rsrch Svc Inc
Attn E Carr
1745 Jefferson Davis Hwy Ste 500
Arlington VA 22202-3402

US Army Rsrch Ofc
Attn AMSRL-RO-D JCI Chang
Attn AMSRL-RO-EN W Bach
PO Box 12211
Research Triangle Park NC 27709

Distribution (cont'd)

US Army Soldier & Biol Chem Cmnd Dir of Rsrch & Techlgy Dirctrt
Attn SMCCR-RS I G Resnick
Aberdeen Proving Ground MD 21010-5423

TECOM
Attn AMSTE-CL
Aberdeen Proving Ground MD 21005-5057

US Army Rsrch Lab
Attn AMSRL-DD J M Miller
Attn AMSRL-CI-AI-A Mail & Records Mgmt

US Army Rsrch Lab (cont'd)
Attn AMSRL-CI-AP Techl Pub (3 copies)
Attn AMSRL-CI-LL Techl Lib (3 copies)
Attn AMSRL-IS-EP P Gillespie
Attn AMSRL-SE J Pellegrino
Attn AMSRL-SE-SA J Eicke
Attn AMSRL-SE-SE M Vrabel (5 copies)
Adelphi MD 20783-1197

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2000	Final, 1 Oct 98 to 30 Sept 99

4. TITLE AND SUBTITLE A Target-Tracking Algorithm

5. FUNDING NUMBERS
DA PR: AH16
PE: 62120A

6. AUTHOR(S) Michael J. Vrabel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
Attn: AMSRL-SE-SE email: vrabel@arl.mil
2800 Powder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-158

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AMCOM*
Redstone Arsenal, AL 35898-5240

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES *Also sponsored by the U.S. Army Research Laboratory.
ARL PR: 9NE4MM
AMS code: 622120.H16

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
A computationally fast algorithm for tracking targets in a sequence of scenes is described. The algorithm is based on a variation of template matching.

14. SUBJECT TERMS
Automatic target recognition, template matching, target tracking

15. NUMBER OF PAGES
41

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102