A Comparison of Pt. Grey Research's Digiclops and Videre Design's Small Vision System for Sensing on the Hyperion Robot

Chris Urmson, November 2000

Introduction

The Hyperion robot will perform a 24 hour traverse of unstructured, lunar analogous terrain in July 2001. To perform this task, the robot will rely on stereo vision to detect obstacles larger than 20cm at a range of 2-5 meters. Two stereo systems were considered- Digiclops, a trinocular stereo system developed by Pt. Grey Research, and the Small Vsion System developed by SRI and marketed through Videre Desgin. The JPL stereo library was ruled out due to concerns regarding the calibration process and the belief that it is too general purpose to be fast enough for our application.

A standard timing procedure was developed and the quality of the stereo data output was qualitatively compared.

Test Setup

The test machine was a Pentium III Celeron 600MHz (128KB cache) with 64MB of memory running RedHat Linux 7.0 (Kernel 2.2.16 patched with IEEE1394 code from Sourceforge CVS, Oct 27, 2000). The computer used an OHCI compliant IEEE1394 adapter card (a SIIG Card with a Texas Instruments TSB12LV23 chipset).

The Digiclops was setup as the only device on the IEEE1394 bus. Pt. Grey's alpha linux frame grabber drivers were used.

Testing of the SVS system used a pair of Sony DFW-VL500 IEEE1394 Cameras with a 15cm baseline. The libdc1394 capture and control library was used to control the cameras.

Both systems were configured to produce 320x240 disparity maps with a search of 32 disparity levels.

All timing tests were performed with X running, but no images or output were displayed to the screen within the timing loops.

Speed

When comparing the two systems, speed (or throughput) is the parameter that most differentiates them. Table 1 shows average times taken for various steps in the process.

	SVS with IEEE1394 Cameras (ms) (over 100 samples)	Digiclops (ms) (over 20 samples)
Image capture	104.5	122.8
Warp	10.7	89.4
Stereo	48.6	305.3
Whole Cycle	**163.8**	**517.5**

Table 1. Comparison of SVS and Digiclops time performance

Pt. Grey has reported that the digiclops system will capture images at 16Hz, but the driver provided was not able to achieve these rates (when the CPU was also burdened with stereo calculations).

When pushed to turn over quickly, it the Digiclops grabbing interface hangs frequently. To get reasonable results (without having the digiclops interface crash) it was necessary to limit the trial length to only 20 samples. The IEEE1394 camera interface was considerably more stable (at a frame rate of 15 FPS), and could run for longer than 100 samples.

Higher capture speeds can be achieved with the IEEE1394 cameras, but currently the driver is not stable enough to support them (currently the cameras are set to stream video at 15FPS). By moving to black and white cameras, the capture time should decrease by a factor of 2 (due to a transition from YUV(4:2:2) to 8bit grayscale data).

Quality

Both systems produced “good” disparity data. The Digiclops system produced better output. In disparity images generated by the Digiclops, obstacles appeared sharper, and data seemed to be more reliable at slightly longer ranges than available from the SVS system. By applying some simple filtering, the results were similar enough that it seems unlikely that either system would produce dramatically improved obstacle detection for Hyperion.

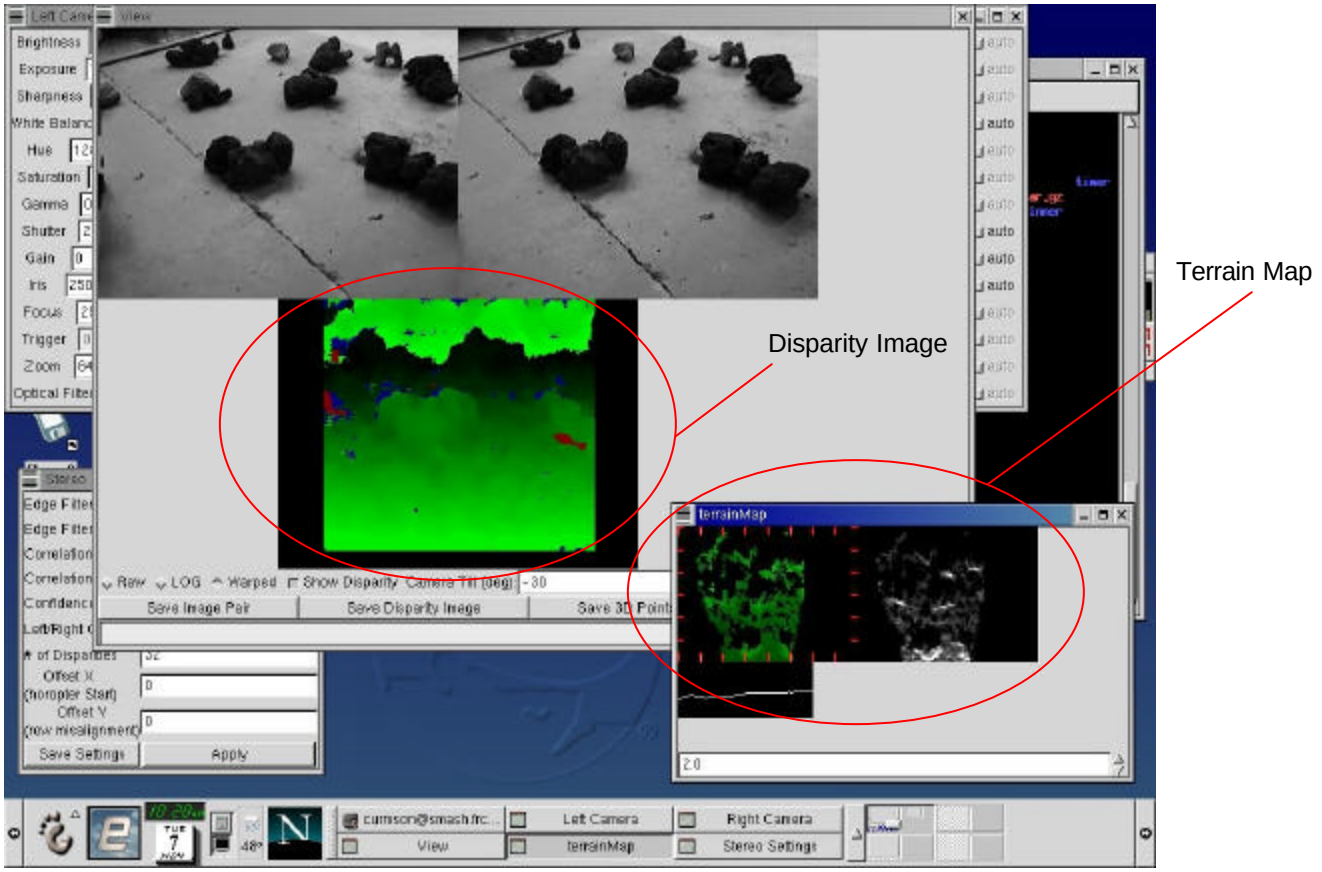

Figure 1. A screen capture showing a disparity image and terrain data from the SVS stereo system.

Figure 1 shows the output of the SVS stereo system. In the disparity image, green represents number of disparities. The sharp change in green intensity is due to roll over in the disparity value (it is a 16bit quanitiy being illustrated by 256 levels of green). Orange/Red represents areas where there was not enough texture. Blue represents areas where correlating from left to right did not return the same match as correlating from right to left.

The left image in the terrain map shows the height of the ground plane, brighter green indicates taller obstacles. Black regions show where no data is available (or the number of points in that region was less than a minimum threshold). The intensity of gray in the right image represents the number of points in each grid cell. The read hash marks represent a distance of 0.5m. The bottom edge of both images is 2m in front of the cameras.

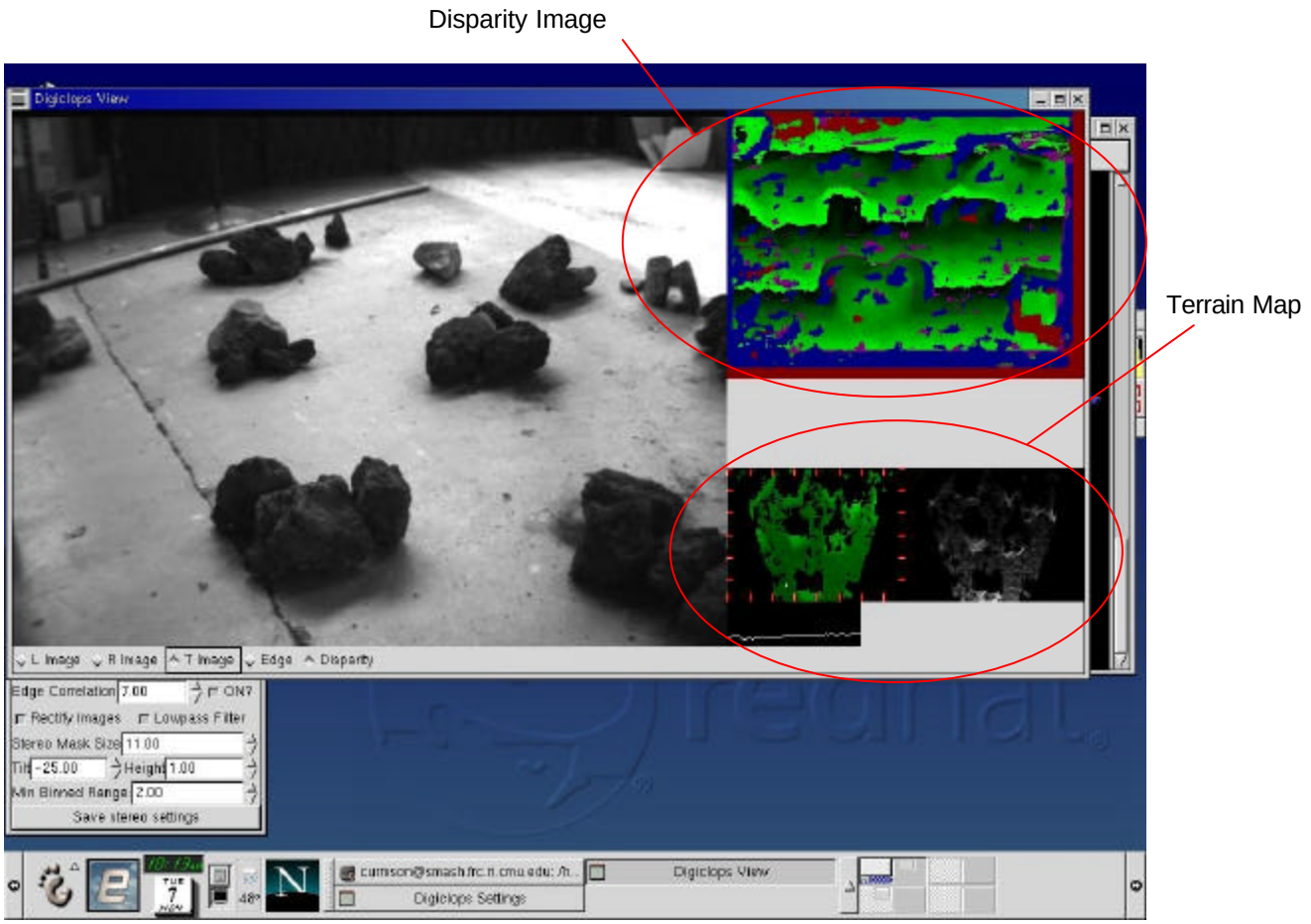

Figure 2. A screen capture showing a disparity image and terrain data from the Digiclops system.

Figure 2 shows the output of the Digiclops stereo system. In the disparity image, green again represents number of disparities. The dramatic difference in the appearance of the disparity maps is due to slightly different techniques for storing the subpixel/pixel disparity values. Orange/Red represents areas where there was not enough texture. Blue represents areas where the correlation was not unique. Purple represents areas where Pt. Grey's proprietary stict subpixel validation has invalidated the match.

The terrain map is the same as described above.

API

Both stereo libraries are setup in an easy to use and straightforward manner. Transitioning between the two API's is also fairly easy, both API's have very similar commands. The Triclops SDK has a more mature feel, and provides more ability to customize the behaviour of the stereo engine. Triclops also provides more features (surface validation, disparity scaling, strict subpixel validation).

Both systems have straightforward calibration procedures. The Digiclops comes precalibrated from the manufacturer. The SVS system uses an algorithm based

on Tsai's algorithm. The user presents the system with an 11"x11" calibration target in 5 poses. From these images, the software is able to extract both extrinsic stereo head parameters and intrinsic camera parameters.

Interfacing

Due to the technique the Digiclops uses to stream data from the camera head to the computer, it is impossible to use additional firewire devices on the same bus. To add a science camera or additional teleoperation cameras would require the addition of another firewire card, or frame grabber.

Neither grabbing system is completely reliable. Unfortunately, the error mode for the digiclops is significantly more difficult to detect. Both systems intermittently fail while attempting to grab images over the IEEE1394 bus. When using the digital cameras, the grabbing mechanism will hang (the call to grab an image fails to return); this can be corrected for by wrapping the grab call in a timeout mechanism, which resets the driver if an image isn't received. This has been tested manually, but not implemented in software.

When using the digiclops capture code, the grabbing mechanism will intermittently lock up and return a single frame repeatedly. The software call is designed to return an error code if the grab fails for some reason, but no error code is returned.

Conclusion

Based on the quantitative and qualitative results described above, Hyperion will be using the SVS stereo system with a pair of Firewire black and white cameras. Though the Pt. Grey system provides slightly better stereo data and has a more mature API, the throughput and interfacing issues cause it to be less sutied to our applicatoin.

Appendix A: Source code for SVS timing application

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#include <sys/mman.h>
#include <libraw1394/raw1394.h>
#include <libdc1394/dc1394_control.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/types.h>
#include <svs.h>
#include "terrain.h"
#include "jpeg_helper.h"

#define NUM_TIMES 50

int main(int argc, char *argv[]) {
  dc1394_cameracapture cameras[3];
  int camIds[2]= {0x302eb,0x302e8};
  int numCameras;
  raw1394handle_t handle;
  nodeid_t* camera_nodes;
  int i;
  unsigned long numPts = 0;
  short dispBuffer[320*240];
  char lImage[320*240],lbuf[320*240];
  char rImage[320*240],rbuf[320*240];
  pt_Cloud * pts;
  terrain_Map  *tmap;

  svsSP stereoPs;
  float   avgwhole=0,   avgcap=0,   avgwarp=0,   avgstereo=0,   avgpc=0,
avgtransform=0;
  struct timeval start, diff;
  struct timeval capture, warp, stereo,getPCloud,transform;
  struct timezone tz;
  tz.tz_minuteswest=0;
  tz.tz_dsttime =0;



  handle = dc1394_create_handle();
  if (handle==NULL) {
    printf("did you insmod the drivers?\n");
    exit(1);
  }
  printf("getting sorted cameras\n");
```

```
  camera_nodes                                                    =
dc1394_get_sorted_camera_nodes(handle,2,camIds,&numCameras,1);
  printf("numCameras: %d, node[0] %d, node[1] %d\n",numCameras,
         camera_nodes[0],camera_nodes[1]);


  dc1394_setup_camera(handle,camera_nodes[0],3,
                      FORMAT_VGA_NONCOMPRESSED,MODE_320x240_YUV422,
                      SPEED_400,FRAMERATE_15,&cameras[0]);
  dc1394_setup_camera(handle,camera_nodes[1],2,
                      FORMAT_VGA_NONCOMPRESSED,MODE_320x240_YUV422,
                      SPEED_400,FRAMERATE_15,&cameras[1]);



  printf("starting timing loop\n");
  svsReadParamFile("good_office.ini",&stereoPs);
  printf("                         start0:                          %d\n",
dc1394_start_iso_transmission(handle,camera_nodes[0]));
  printf("                                                        start1:
%d\n",dc1394_start_iso_transmission(handle,camera_nodes[1]));
  for (i=0;i<NUM_TIMES+1;i++) {
    gettimeofday(&start,&tz);
    dc1394_multi_capture(handle,cameras,numCameras);

    yuv2grey8b((char *)cameras[0].capture_buffer,lImage,320*240);
    yuv2grey8b((char *)cameras[1].capture_buffer,rImage,320*240);
    gettimeofday(&capture,&tz);
    svsWarpImage(lbuf,lImage,svsLEFT,&stereoPs);
    svsWarpImage(rbuf,rImage,svsRIGHT,&stereoPs);
    gettimeofday(&warp,&tz);
    svsCalcStereo(dispBuffer,lImage,rImage,&stereoPs);
    gettimeofday(&stereo,&tz);

    pts = ptCloudfromSVS(dispBuffer, &stereoPs,-15.0,1.0);
    gettimeofday(&getPCloud,&tz);
    tmap = tmapfromPtCloud(pts);
    gettimeofday(&transform,&tz);

    if (i!=0) {
        numPts+=pts->numPts;
        timersub(&capture,&start,&diff);
        avgcap+=diff.tv_usec/(1000.0);
        timersub(&warp,&capture,&diff);
        avgwarp+=diff.tv_usec/(1000.0);
        timersub(&stereo,&warp,&diff);
        avgstereo+=diff.tv_usec/(1000.0);
        timersub(&getPCloud,&stereo,&diff);
        avgpc +=diff.tv_usec/(1000.0);
        timersub(&transform,&getPCloud,&diff);
        avgtransform+=diff.tv_usec/(1000.0);
        timersub(&transform,&start,&diff);
        avgwhole+=diff.tv_usec/(1000.0);
        printf("%d pts: %d bad: %d\n",i, pts->numPts,pts->numBadPts);
    }
  }
```

```
  printf("Stereo timing results for SVS:\n");
  printf("Capture took : %fms\n",avgcap/NUM_TIMES);
  printf("Warp took : %fms\n",avgwarp/NUM_TIMES);
  printf("Stereo took : %fms\n",avgstereo/NUM_TIMES);
  printf("Disparity to pt. Cloud took : %fms\n",avgpc/NUM_TIMES);
  printf("Pt. cloud to bins took : %fms\n",avgtransform/NUM_TIMES);
  printf("# of Pts/Pt. Cloud: %u\n",numPts/NUM_TIMES);
  printf("----------------------\nWhole       cycle       took       :
%fms\n",avgwhole/NUM_TIMES);


  dc1394_stop_iso_transmission(handle,camera_nodes[0]);
  dc1394_stop_iso_transmission(handle,camera_nodes[1]);
  dc1394_release_camera(handle,&cameras[0]);
  dc1394_release_camera(handle,&cameras[1]);
  raw1394_destroy_handle(handle);
  exit(0);


}
```

Appendix B: Source code for Digiclops timing application

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#include <sys/mman.h>
#include <libraw1394/raw1394.h>
//#include <libdc1394/dc1394_control.h>

#include <netinet/in.h>
#include <sys/time.h>
#include <sys/types.h>
//#include <svs.h>
#include <digi-cam.h>
#include <triclops.h>
#include "terrain.h"
#define ERROR(FMT...) { fprintf( stderr, FMT ); exit( 1 ); }

#define NUM_TIMES 20

int main(int argc, char *argv[]) {
  int ndev;
  int i,j;
  unsigned long numPts = 0;
  unsigned char *buffer=0;
  int max_delay =2;
  short dispBuffer[320*240];
  unsigned char bufferA[640*480];
  unsigned char bufferB[640*480];
  unsigned char bufferC[640*480];
  float   avgwhole=0,   avgcap=0,   avgwarp=0,   avgstereo=0,   avgpc=0,
avgtransform=0;
  struct timeval start, diff;
  struct timeval capture, warp, stereo,getPCloud,transform;
  struct timezone tz;
  pt_Cloud * pts;
  terrain_Map  *tmap;
  DigiCamHandle digi;

  TriclopsImage16 dispImage;
  TriclopsContext context;
  TriclopsInput inputData;

  tz.tz_minuteswest=0;
  tz.tz_dsttime =0;

  if (digi_cam_init()<0) {
    ERROR("Couldn't initialize 1394\n")
      }
```

```
  ndev = digi_cam_num_device();
  if (ndev <1) {
    ERROR("no digiclops\n");
  }
  printf("we got a device\n");

  if (digi_cam_get_device(&digi,1)!=1) {
    ERROR("couldn't get a handle\n");
  }

  digi_cam_alloc_buffer(digi);

  printf("loading context\n");
  if
(triclopsGetDefaultContextFromFile(&context,"digiclops0310014.cal")!=ok
) {
    ERROR("couldn't get default context from file\n");
  }
  printf("context loaded\n");

  triclopsSetResolution(context,240,320);
  triclopsSetDisparity(context,0,32);
  triclopsSetSubpixelInterpolation(context,1);
  inputData.inputType = TriInp_RGB;
  inputData.nrows = 480;
  inputData.ncols = 640;
  inputData.rowinc = 640;
  inputData.u.rgb.red = bufferA;
  inputData.u.rgb.green = bufferB;
  inputData.u.rgb.blue = bufferC;


  digi_cam_grab(digi,1);
  for (j=0;j<NUM_TIMES+1;j++) {
    gettimeofday(&start,&tz);
    if (digi_cam_get_buffer(digi,&buffer,max_delay)<0 || buffer==0) {
      ERROR("couldn't grab a buffer\n");
    }
    for (i=0;i<640*480;i++) {
      bufferA[i]=buffer[i*3];
      bufferB[i]=buffer[i*3+1];
      bufferC[i]=buffer[i*3+2];
    }
    gettimeofday(&capture,&tz);

    triclopsPreprocess(context,&inputData);
    gettimeofday(&warp,&tz);
    triclopsStereo(context);
    gettimeofday(&stereo,&tz);

triclopsGetImage16(context,TriImg_DISPARITY,TriCam_REFERENCE,&dispImage
);
    pts = ptCloudfromPG(dispImage.data,context,-15.0,1.0);
    gettimeofday(&getPCloud,&tz);
    tmap = tmapfromPtCloud(pts);
    gettimeofday(&transform,&tz);
    if (i>0) {
```

```
        numPts+=pts->numPts;
        timersub(&capture,&start,&diff);
        avgcap+=diff.tv_sec*1000.0 + diff.tv_usec/(1000.0);
        timersub(&warp,&capture,&diff);
        avgwarp+=diff.tv_usec/(1000.0);
        timersub(&stereo,&warp,&diff);
        avgstereo+=diff.tv_usec/(1000.0);
        timersub(&getPCloud,&stereo,&diff);
        avgpc +=diff.tv_usec/(1000.0);
        timersub(&transform,&getPCloud,&diff);
        avgtransform+=diff.tv_usec/(1000.0);
        timersub(&transform,&start,&diff);
        avgwhole+=diff.tv_sec*1000.0+diff.tv_usec/(1000.0);
        printf("%d\n",j);
    }
  }
  digi_cam_grab(digi,0);


  printf("Stereo timing results for Digiclops:\n");
  printf("Capture took : %fms\n",avgcap/NUM_TIMES);
  printf("Warp took : %fms\n",avgwarp/NUM_TIMES);
  printf("Stereo took : %fms\n",avgstereo/NUM_TIMES);
  printf("Disparity to pt. Cloud took : %fms\n",avgpc/NUM_TIMES);
  printf("Pt. cloud to bins took : %fms\n",avgtransform/NUM_TIMES);
  printf("# of Pts/Pt. Cloud: %u\n",numPts/NUM_TIMES);
 printf("----------------------\nWhole       cycle       took       :
%fms\n",avgwhole/NUM_TIMES);
  digi_cam_free_buffer(digi);
  digi_cam_term();
  exit(0);


}
```

Appendix C: Source code for terrain.h

```
#ifndef __TERRAIN_H
#define __TERRAIN_H
#define TRICLOPS
#define VIDER
#include <stdio.h>
#ifdef VIDER
#include <svs.h>
#endif
#ifdef TRICLOPS
#include <triclops.h>
#endif


#include <math.h>
#include <stdlib.h>



#define TER_WIDTH 320
#define TER_HEIGHT 240
#define IMAGE_SIZE TER_WIDTH*TER_HEIGHT
typedef struct _ptCloud {
  float X[IMAGE_SIZE];
  float Y[IMAGE_SIZE];
  float Z[IMAGE_SIZE];
  int numPts;
  int numBadPts;
} pt_Cloud;

#define TER_MINX -2
#define TER_MAXX +2
#define TER_MINY 0
#define TER_MAXY 3
#define TER_DEFY 2
#define TER_BINS_METER 20

#define TER_DIMX ((TER_MAXX-TER_MINX)*TER_BINS_METER)
#define TER_DIMY ((TER_MAXY-TER_MINY)*TER_BINS_METER)




typedef struct _terrainMap {
  short numSamples[TER_DIMX*TER_DIMY];
  float height[TER_DIMX*TER_DIMY];
  int numOutofRange;
  int numInRange;
  float minHeight,maxHeight;
  int minSamples,maxSamples;
} terrain_Map;
#ifdef VIDER
pt_Cloud  *  ptCloudfromSVS(unsigned  short  *disp,  svsSP  *param,float
angle,float vert_offset);
#endif
```

```
#ifdef TRICLOPS
pt_Cloud * ptCloudfromPG(unsigned short *disp, TriclopsContext context,
float angle, float vert_offset);
#endif

terrain_Map * tmapfromPtCloud(pt_Cloud * pts);
terrain_Map * tmapfromPtCloudArbY(pt_Cloud * pts,float tminy);
#endif
```

Appendix D: Source code for terrain.c

```
#include "terrain.h"
#include <math.h>

#ifdef TRICLOPS
pt_Cloud * ptCloudfromPG(unsigned short *disp, TriclopsContext context,
float angle, float vert_offset) {
  pt_Cloud * pts;
  int h,i,j,w,index,outIndex;
  float c,s;
  float ztemp,ytemp,x,y,z;
  int numBad =0; //number of bad points;

  pts = malloc(sizeof(pt_Cloud));
  if (pts==NULL) {
    printf("(%s:%d) unable to allocate space for pt. cloud\n",
         __FILE__,__LINE__);
    return NULL;
  }
  c = cos(angle*M_PI/180.0);
  s = sin(angle*M_PI/180.0);
  w = 320;
  h = 240;
  outIndex=0;
  for (j=0;j<h;j++) {
    for (i=0;i<w;i++) {
      index=j*w+i;
      if (disp[index]<0xFF00) {
      triclopsRCD16ToXYZ(context,j,i,disp[index],&x,&y,&z);
      /*from svsReconstuct
        z pts out axis of left camera
        y pts vertically down in camera frame
        x pts right in camera frame
      and units are mm so we convert them to meters here*/
      ytemp = (z*c+y*s); //transform to world coords
      ztemp = (z*s-y*c) +vert_offset;
      if (finite(x) && finite(ytemp) && finite(ztemp)) {
        pts->X[outIndex]=x;
        pts->Y[outIndex]=ytemp;
        pts->Z[outIndex]=ztemp;
        outIndex++;
      } else {
        numBad++;
      }
      } else {
      numBad++;
      }
    }
  }
  pts->numPts = outIndex;
  pts->numBadPts = numBad;
  //  printf("numBadPts==%d, numPts==%d\n",numBad,outIndex);
  return pts;
}
```

```
#endif

#ifdef VIDER
pt_Cloud  *  ptCloudfromSVS(unsigned  short  *disp,  svsSP  *params,float
angle,float vert_offset) {
  pt_Cloud * pts;
  int h,i,j,w,index,outIndex;
  int imw,imh;//image width and height
  float c,s;
  float ztemp,ytemp,x,y,z;
  int numBad =0; //number of bad points;
  pts = malloc(sizeof(pt_Cloud));
  if (pts==NULL) {
    printf("(%s:%d) unable to allocate space for pt. cloud\n",
         __FILE__,__LINE__);
    return NULL;
  }
  c = cos(angle*M_PI/180.0);
  s = sin(angle*M_PI/180.0);
  w = RESWIDTH(params);//width of disparity data in image
  h = RESLEN(params);//height of disparity data in image
  outIndex=0;
  imw= params->width;
  imh = params->height;
  for (j=0;j<h;j++) {
    for (i=0;i<w;i++) {
      index=j*imw+i;
      if (disp[index]!=0xffff && disp[index]!=0xfffe) {
      svsReconstruct3DFast(&x,&y,&z,i,j,disp[index],params);
      /*from svsReconstuct
        z pts out axis of left camera
        y pts vertically down in camera frame
        x pts right in camera frame
      and units are mm so we convert them to meters here*/
      ytemp = (z*c+y*s)/1000.0; //transform to world coords
      ztemp = (z*s-y*c)/1000.0 +vert_offset;
      if (finite(x) && finite(ytemp) && finite(ztemp)) {
        pts->X[outIndex]=x/1000.0;
        pts->Y[outIndex]=ytemp;
        pts->Z[outIndex]=ztemp;
        outIndex++;
      } else {
        numBad++;
      }
      } else {
      numBad++;
      }
    }
  }
  pts->numPts = outIndex;
  pts->numBadPts = numBad;
  return pts;
}
#endif

terrain_Map * tmapfromPtCloud(pt_Cloud * pts) {
  int i;
```

```
terrain_Map *tmap;
int cordx, cordy;
int numOutofRange=0;
int numInRange=0;
float x,y,z;
float tminy=TER_DEFY;
tmap = malloc(sizeof(terrain_Map));
if (tmap==NULL) {
  printf("(%s:%d) unable to allocate space for terrain map\n",
       __FILE__,__LINE__);
  return NULL;
}
tmap->minHeight =999;
tmap->maxHeight =-999;
tmap->minSamples = 999;
tmap->maxSamples = -1;


for (i=0;i<TER_DIMX*TER_DIMY;i++) {
  tmap->numSamples[i]=0;
  tmap->height[i]=0;
}

//  printf("num pts is: %d\n",pts->numPts);
for (i=0;i<pts->numPts;i++) {
  x= pts->X[i];
  y = pts->Y[i];
  z = pts->Z[i];

  if (x>TER_MINX && x<TER_MAXX && y>tminy && y<(tminy+TER_MAXY)) {
    cordx = floor(x*TER_BINS_METER)-TER_MINX*TER_BINS_METER;
    cordy = floor(y*TER_BINS_METER)-tminy*TER_BINS_METER;
    tmap->numSamples[cordx+cordy*TER_DIMX]++;
    tmap->height[cordx+cordy*TER_DIMX]+=z;
    numInRange++;
  } else {
    numOutofRange++;
  }
}
//  printf("numinrange: %d, numout: %d\n",numInRange,numOutofRange);

for (i=0;i<TER_DIMX*TER_DIMY;i++) {
  if (tmap->numSamples[i]>0) {
    tmap->height[i]/=tmap->numSamples[i];

    // collect some basic statistics on cells with data
    if (tmap->height[i]>tmap->maxHeight) {
    tmap->maxHeight= tmap->height[i];
    }

    if (tmap->height[i]<tmap->minHeight) {
    tmap->minHeight=tmap->height[i];
    }
    if (tmap->numSamples[i]>tmap->maxSamples) {
    tmap->maxSamples = tmap->numSamples[i];
    }
    if (tmap->numSamples[i]<tmap->minSamples) {
```

```
      tmap->minSamples = tmap->numSamples[i];
      }

    }
  }

  tmap->numInRange=numInRange;
  tmap->numOutofRange=numOutofRange;
  return tmap;
}


terrain_Map * tmapfromPtCloudArbY(pt_Cloud * pts,float tminy) {
  int i;
  terrain_Map *tmap;
  int cordx, cordy;
  int numOutofRange=0;
  int numInRange=0;
  float x,y,z;
  tmap = malloc(sizeof(terrain_Map));
  if (tmap==NULL) {
    printf("(%s:%d) unable to allocate space for terrain map\n",
         __FILE__,__LINE__);
    return NULL;
  }
  tmap->minHeight =999;
  tmap->maxHeight =-999;
  tmap->minSamples = 999;
  tmap->maxSamples = -1;


  for (i=0;i<TER_DIMX*TER_DIMY;i++) {
    tmap->numSamples[i]=0;
    tmap->height[i]=0;
  }

  //  printf("num pts is: %d\n",pts->numPts);
  for (i=0;i<pts->numPts;i++) {
    x= pts->X[i];
    y = pts->Y[i];
    z = pts->Z[i];

    if (x>TER_MINX && x<TER_MAXX && y>tminy && y<(tminy+TER_MAXY)) {
      cordx = floor(x*TER_BINS_METER)-TER_MINX*TER_BINS_METER;
      cordy = floor(y*TER_BINS_METER)-tminy*TER_BINS_METER;
      tmap->numSamples[cordx+cordy*TER_DIMX]++;
      tmap->height[cordx+cordy*TER_DIMX]+=z;
      numInRange++;
    } else {
      numOutofRange++;
    }
  }
  //  printf("numinrange: %d, numout: %d\n",numInRange,numOutofRange);

  for (i=0;i<TER_DIMX*TER_DIMY;i++) {
    if (tmap->numSamples[i]>0) {
      tmap->height[i]/=tmap->numSamples[i];
```

```
      // collect some basic statistics on cells with data
      if (tmap->height[i]>tmap->maxHeight) {
      tmap->maxHeight= tmap->height[i];
      }

      if (tmap->height[i]<tmap->minHeight) {
      tmap->minHeight=tmap->height[i];
      }
      if (tmap->numSamples[i]>tmap->maxSamples) {
      tmap->maxSamples = tmap->numSamples[i];
      }
      if (tmap->numSamples[i]<tmap->minSamples) {
      tmap->minSamples = tmap->numSamples[i];
      }

    }
  }

  tmap->numInRange=numInRange;
  tmap->numOutofRange=numOutofRange;
  return tmap;
}
```