MOTION CAPTURE AND HUMANOIDS

Andrew Waterreus
Robotics and Motion Laboratory,
Dept. of Mechanical Engineering
San Antonio, TX, USA 78249
ajwhersh@gmail.com

ABSTRACT

The generation of new custom motions with robots, especially humanoids, requires a significant amount of time and is quite challenging, especially for those new to programming, or the operation of robotic systems. In order to reduce the amount of time required, as well as significantly make this process easier for beginners, a motion capture system could be utilized to track the motion of a human occupant, and generate data that could then be uploaded to the robot as a new custom motion.

The proper utilization of a DARWIN OP2 robot and how to develop new motions needed to be learnt in order to realize the challenge needing to be overcome. Next, simulation and animation files were provided to allow for faster testing of the processed data acquired from a motion capture system. These took a significant amount of time to learn to use due to lack of prior programming knowledge, and unfamiliarity with the structure of the particular codes. Data was acquired from the motion capture system about a simple movement of the right arm which the robot would have been capable of replicating, but the data was not processed into a form that was able to be tested by the animation code before the semester ended. Thus, more work in the future needs to be performed in order to see that this experiment comes to fruition.

1. NOMENCLATURE

RAM – Robotics and Motion
DOF – Degrees of Freedom

2. INTRODUCTION

While learning to operate a DARWIN OP2 humanoid robot, the difficulty of generating new custom robotic motions was realized. A significant amount of time could potentially be saved by developing a method of processing the extensive data generated by a motion capture system, into a form that would be usable by the robot. Luckily, C programs had already been developed to generate custom data and an accompanying humanoid animation by another student. This would allow for the processed data from the motion capture system to be rapidly tested on an animation, rather than needing to constantly upload and run each test on the physical robot. Although the human body is much more complex than the DARWIN OP2 robot could possibly represent, the idea was chosen to be pursued, as long as motions were kept simple.

3. METHODS

First a position in the RAM Lab at UTSA needed to be acquired as it possessed the required materials and equipment necessary to perform this experiment, as well as mentorship in the form of Dr. Pranav Bhounsule and other more advanced students. This was done early in the semester so as to become familiar with the equipment and the other lab members. The experiment would be performed with a DARWIN OP2 humanoid robot supplied by the RAM lab, C code files for both simulation and animation generation of a humanoid robot supplied by another student member in the lab, and lastly, the VICON motion capture system available in Dr. Amir Jafari's side of the lab. The code files and images of the equipment can be found in Appendix sections A and B respectively.

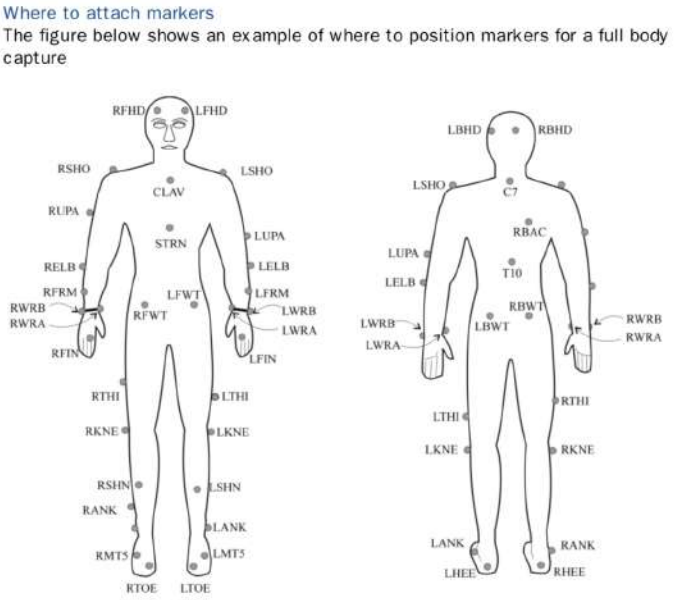
In performing the experiment, the operation of the DARWIN OP2 robot first needed to become familiar, so time was spent in the lab learning how to modify its C++ files in order to make it perform desired tasks, such as different physical or vocal animations. The robot did come with a small booklet that provided introductory le During this portion of the project, an opportunity to represent the UTSA RAM lab on Live TV by modifying the DARWIN robot to perform a simple introduction and greeting at the opening of the San Antonio DoSeum's new exhibit, "Science Fiction, Future Science," was presented. This opportunity provided excellent motivation to quickly learn how to modify voice animation files, and to create unique body motions.

Next, time needed to be spent on learning to properly utilize the humanoid simulation and animation files provided by the graduate student Robert Brothers. This required a significant amount of time due to lack of prior coding knowledge and needing to decipher a skilled programmer's codes. The codes were not written for a beginner level programmer so they did not have many descriptive comments to provide explanations of each line. The simulation and animation codes, as well as images of the animation can be viewed in Appendix sections A and B respectively.

Once a basic understanding of the code files was obtained, the proper utilization of the VICON motion capture system needed to be learned. Currently, only a few students working under Dr. Jafari know how to use it, so the grad student Christian Wahrmund was consulted, as was advised by Dr. Bhounsule. Christian provided a satisfactory explanation of how the system is able to acquire usable data about motion within the system during utilization. First the VICON Nexus, or the system's software was activated on Christian's computer. Christian performed this while I set the system up. Two tripods with high quality cameras were set up among the already mounted cameras in order to bring the total number of cameras to eight. Extra care was taken during this step, as the equipment for the motion capture system alone is worth over $40,000.

Next, the system was calibrated in order for all of the cameras to know where they were with respect to each other and a coordinate origin point generated by Christian. This was done by utilizing a wand with four LEDs and algorithms that come with the VICON software that uses the known distances and relations of the LEDs to analyze the varying images of the cameras to develop relations between each camera. Even a tiny bump on one of the cameras would upset these relationships and cause the entire system to require recalibration, as was experienced once.

The cameras operate by tracking small reflective silver markers placed on the user's body. The correct placing of the markers can be seen in the image below.



VICON system marker placement

The cameras work by scanning for the specific reflectivity of the markers and generating 3-D coordinates for each of them within the system. For the project, it was determined that it would be best to only gather data on the motion of my right arm, as the system generates a significant amount of data and a "motion" that the DARWIN robot would be capable of performing needed to be done. The robot has only 20 DOF, while the human body has 244, although many are utilized in small actions like bending of fingers, which the robot does not even have. In the recorded motion, only the shoulder and elbow joints were rotated, as the robot's arm has 4 DOF here, the same as a human arm. Thus, it should be capable of replicating the motion within reasonable limits. The VICON Nexus saves all of the data required for any VICON Nexus software to digitally repeat the motion.

Finally, the motion capture data needed to be processed to convert the Cartesian coordinates of the markers into rotation angles of the shoulder and elbow which could be used in the simulation, as well as the robot itself. Unfortunately, this was not completed before the Fall 2017 semester ended, and so will need to be performed at a later date.

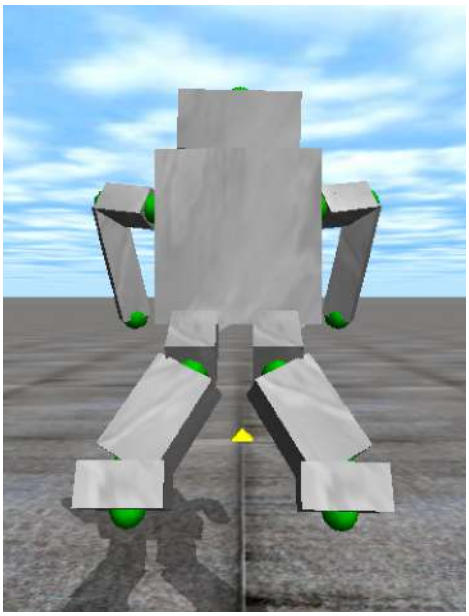
4. RESULTS

The simulation file, when run, generates a significant amount of data in the form of coordinate and rotation values for the entire body of the simple humanoid model generated by the animation file, as can be seen in this image.

```
Andrews-MacBook-Pro-2:darwin_animation andrewwaterreus$ ./simulateFwd
0.000000 0.000000 0.000000 0.005000 0.000000 0.172700
0.005000 -0.088232 0.122200 0.005000 -0.088232 0.106200
0.005000 0.082000 0.122200 0.005000 0.082000 0.106200
0.005000 -0.088232 0.106200 0.005000 -0.148232 0.106200
0.005000 0.082000 0.106200 0.005000 0.142000 0.106200
0.005000 -0.148232 0.106200 -0.122040 -0.170634 0.106200
0.005000 0.142000 0.106200 -0.122040 0.164402 0.106200
0.000000 -0.043232 0.000000 0.000000 -0.043232 0.000000
0.000000 0.037000 0.000000 0.000000 0.037000 0.000000
0.000000 -0.043232 0.000000 0.000000 -0.043232 0.000000
0.000000 0.037000 0.000000 0.000000 0.037000 0.000000
0.000000 -0.043232 0.000000 0.062692 -0.024101 -0.065975
0.000000 0.037000 0.000000 0.062692 0.017870 -0.065975
0.062692 -0.024101 -0.065975 0.066333 -0.085507 -0.135725
0.062692 0.017870 -0.065975 0.066333 0.079276 -0.135725
0.066333 -0.085507 -0.135725 0.066333 -0.085507 -0.135725
0.066333 0.079276 -0.135725 0.066333 0.079276 -0.135725
0.066333 -0.085507 -0.135725 0.061643 -0.102302 -0.164329
0.066333 0.079276 -0.135725 0.061643 0.096070 -0.164329
0.005000 0.000000 0.172700 0.005000 0.000000 0.172700
0.005000 0.000000 0.172700 0.020335 0.000000 0.229931
0.000000 0.000000 0.000000 0.005000 0.000000 0.172700
0.005000 -0.088232 0.122200 0.005698 -0.088232 0.106215
0.005000 0.082000 0.122200 0.005698 0.082000 0.106215
0.005698 -0.088232 0.106215 0.005652 -0.148223 0.107261
0.005698 0.082000 0.106215 0.005652 0.141991 0.107261
0.005652 -0.148223 0.107261 -0.121583 -0.168846 0.102067
0.005652 0.141991 0.107261 -0.121583 0.162614 0.102067
0.000000 -0.043232 0.000000 0.000000 -0.043232 0.000000
0.000000 0.037000 0.000000 0.000000 0.037000 0.000000
-0.000000 -0.043232 0.000000 -0.000000 -0.043232 0.000000
-0.000000 0.037000 0.000000 -0.000000 0.037000 0.000000
0.000000 -0.043232 -0.000000 0.063573 -0.024520 -0.065248
```

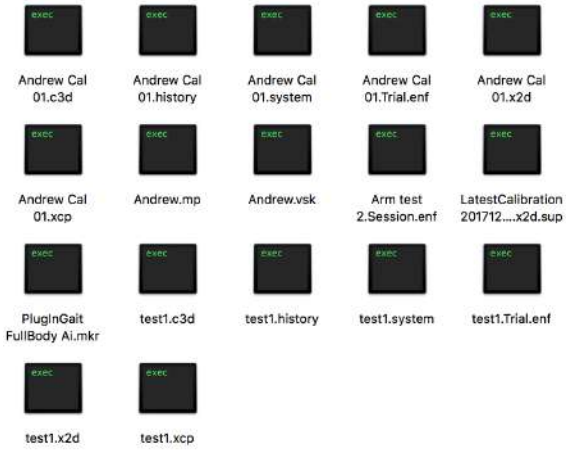
Simulation file data

When the animation file is run it generates a basic humanoid figure based off of the simulation files data. Depending on the data, the figure can assume a large variety of positions; an example can be seen in the following image.



Animation based on simulation file data

After the motion capture system was utilized, the data for the whole test was saved so that it could be opened on, and digitally repeated from any complete VICON software. Unfortunately, the charts of the Cartesian coordinates for each marker were only able to currently be viewed on the software and images of them were not taken at the time when the system was being used. A figure of the data file showing how much data was generated can be seen in the following image though, as well as the large variety of file formats, observed through the subscripts at the end of each file's name.



Arm Test 2 Folder image

5. DISCUSSION

The simulation code file generates a large list of coordinate and angular rotation values that are then used by the animation file in generating the humanoid image. This experiment would test the possibility of utilizing the data from the VICON system, after some filtering and other post processing, as a potential substitute for the simulation files data. Due to running out of time, this possibility was not fully tested unfortunately.

The data collected from the motion capture test needs to be able to be viewable from software other than VICON, so I needed to search for software packages that could open the various file formats, which was a task not previously expected. A potential method suggested by Dr. Bhounsule was software packages available for download on MATLAB.

Although data was able to be obtained about the motion of an arm, and the basic operation and utilization of the DARWIN OP2 robot was able to be learnt, the full experiment was not able to be completed. Limitations in my approach that could be the cause of this may have been that I started with almost zero knowledge of programming or the utilization of the required equipment. Due to my lack of familiarity with the VICON software, I had not expected the format of the data it would produce, and how many different file formats it would be generated as.

6. CONCLUSION AND FUTURE WORK

The operation and utilization of the UTSA RAM Lab's DARWIN OP2 was learnt and during this process, the challenge of generating new, custom motions with robots, especially humanoids, was realized. The possibility of using motion capture, to gather data about a human test subject, and then using that data to more rapidly generate the custom motions was presented. In order to more easily start this experiment, C code files were provided that could first generate custom data in a form that could potentially be similar to the data generated by a motion capture system; then the data would be used to generate a moving humanoid figure through an animation code. The data eventually obtained by the motion capture system needed some filtering and post processing, as the VICON system rapidly generates a significant amount of data. The system generated 30.1 MB of data through less than ten seconds of motion capture of only seven markers on one arm. This is larger than most full textbook pdf files. Limitations that delayed progress through this experiment would be lack of prior programming knowledge, as well as no familiarity with the equipment.

In the future, a proper method of rapidly filtering the VICON data into a form usable by the animation file, and potentially the DARWIN robot, might be found. The simulation and animation codes could also be more fully understood, as this was a large source of delay. Potentially the codes could be modified to include descriptive comments so that other beginners could more rapidly use them.

ACKNOWLEDGMENTS

I'd like to acknowledge Dr. Pranav Bhounsule and Dr. Amir Jafari for their mentorship and the ability to use their equipment, as well as a position in the RAM Lab.

I'd also like to acknowledge Robert Brothers and Christian Wahrmund for their assistance in performing the experiment.

APPENDIX

A. Code

A.1 Simulation.c code

```
#include <stdio.h>
#include <math.h>
#include "params.h"

#include <string.h>
#include "le_ludecomp.c"
#include "useful.c"

int save_date_for_animation();
FILE *fid;
int HEAD_HEIGHT = 1; //If HEAD_YAW is set to (Lf+L5+L4+Lty+Lh) in MATLAB then do this, we assume that the height is


int main()
{

    double t=0, q[NQ] = {0}, u[NU] = {0}; q[NQ-1] = 1;
    int i;


    double tend = 10*60;
    double dt = 0.1;

    fid = fopen("data.txt","w");


  double temp_pos[3];

  sdinit(); sdprinterr(stderr);
  //sdstate(t,q,u); sdprinterr(stderr);

  ///////// modify the head joint location %%%
  if (HEAD_HEIGHT==1) //Takes the height to (Lf+L5+L4+Lty+Lh/2) instead of (Lf+L5+L4+Lty+Lh) (in MATLAB)
  {
     sdgetbtj(HEAD_YAW,temp_pos); temp_pos[2] = temp_pos[2]-Lh/2;
     sdbtj(HEAD_YAW,temp_pos);

     sdgetitj(HEAD_PITCH,temp_pos); temp_pos[2] = temp_pos[2]-Lh/2;
     sditj(HEAD_PITCH,temp_pos);
     sdgetbtj(HEAD_PITCH,temp_pos); temp_pos[2] = temp_pos[2]-Lh/2;
     sdbtj(HEAD_PITCH,temp_pos);
  }


  sdinit(); sdprinterr(stderr);
  sdstate(t,q,u); sdprinterr(stderr);

    while (1)
    {
```

```
            int k=0;
            for (i=6;i<NQ-1;i++)
            {
        double tt = 0.1*t;

        //if (i==BT2 || i==BS2 || i== BF2 || i==BSR || i==BSL)
        //{//do nothing
        //}
        //else
        //{
           //printf("%d \n",k);
           //q[i] = joint_limits_rad[k][1] + (sin(tt)+1)*( (joint_limits_rad[k][0]-joint_limits_rad[k][1])/2.0);
           q[i] =  0.5*(-sin(tt)+1)*joint_limits_rad[k][0] + 0.5*(sin(tt)+1)*joint_limits_rad[k][1];
           //q[i] = joint_limits_rad[k][0] ;
           //q[i] = joint_limits_rad[k][1] ;
           //q[i] = (joint_limits_rad[k][0]+joint_limits_rad[k][1])/2.0;


           //if (k==LSS || k==RSS)
              //printf("%f ",q[k]);
           k +=1;


        //}
        //printf("\n");
            }

            sdstate(t,q,u); sdprinterr(stderr);

      //save data to a file
      save_data_for_animation();

            if (t>tend)
                  break;

            t += dt;

      }

      //close file
      fclose(fid);

   // check for errors
   sdprinterr(stderr);

   return 0;
}


void
sduforce(double t, double *q, double *u)
{}

int save_data_for_animation()
{
   double pos1[3],pos2[3];
```

```
int info[50], slider[6];
int inb_body_no,outb_body_no;
double inb_to_joint[3], body_to_joint[3];
double pos_end_effector[3] = {0};
int ii,jj,i;



//Handle the torso first
sdgetbtj(TORSO,body_to_joint); sdprinterr(stderr);
sdpos(TORSO,body_to_joint,pos1); sdprinterr(stderr);
sdgetitj(HEAD_YAW,inb_to_joint); sdprinterr(stderr);
sdpos(TORSO,inb_to_joint,pos2);
fprintf(fid,"%f %f %f ",pos1[0],pos1[1],pos1[2]);
fprintf(fid,"%f %f %f ",pos2[0],pos2[1],pos2[2]);

printf("%f %f %f ",pos1[0],pos1[1],pos1[2]);
printf("%f %f %f ",pos2[0],pos2[1],pos2[2]);
printf("\n");

// All joints except the TORSO joint
for (ii=1;ii<NBOD;ii++) //cycle through info file
{
   //      (body=outb_body_no) - body_to_joint - (joint=outb_body_no) - inb_to_joint - (body=inb_body_no)
   // sdjnt(joint,info,slider);
   // where info[2] info[3] is
   // inboard outboard (from _i file)

   //This loop finds the start position of all joints except the TORSO
   for (jj=1;jj<NBOD;jj++)
   {
      sdjnt(jj,info,slider); sdprinterr(stderr);
      //inb_body_no = info[2]; //inboard body number = inboard joint number
      outb_body_no = info[3]; //outboard body number = outboard joint number
      if (outb_body_no==ii)
      {
         sdgetbtj(outb_body_no,body_to_joint); sdprinterr(stderr);
         sdpos(outb_body_no,body_to_joint,pos1); sdprinterr(stderr);
      }
   }

   //This find the end position of all joints except the TORSO
   //Need to handle end-effectors differently
   if (ii == R_ELBOW_YAW)
   {
      pos_end_effector[0] = 0; pos_end_effector[1] = -LH; pos_end_effector[2] = 0; //position of end-effector wrt last joint (see
Darwin kinematics figure from ASME paper)
      for (i=0;i<3;i++)
         pos_end_effector[i] += body_to_joint[i]; //position of end-effector wrt to com, as com = 0,0,0
      sdpos(ii,pos_end_effector,pos2);

   }
   else if (ii == L_ELBOW_YAW)
   {
```

```
            pos_end_effector[0] = 0; pos_end_effector[1] = LH; pos_end_effector[2] = 0; //position of end-effector wrt last joint (see Darwin kinematics figure from ASME paper)
            for (i=0;i<3;i++)
               pos_end_effector[i] += body_to_joint[i]; //position of end-effector wrt to com, as com = 0,0,0
            sdpos(ii,pos_end_effector,pos2);
         }
         else if (ii==R_ANKLE_ROLL)
         {
            pos_end_effector[0] = 0; pos_end_effector[1] = 0; pos_end_effector[2] = -LF; //position of end-effector wrt last joint (see Darwin kinematics figure from ASME paper)
            for (i=0;i<3;i++)
               pos_end_effector[i] += body_to_joint[i]; //position of end-effector wrt to com, as com = 0,0,0
            sdpos(ii,pos_end_effector,pos2);
         }
         else if (ii==L_ANKLE_ROLL)
         {
            pos_end_effector[0] = 0; pos_end_effector[1] = 0; pos_end_effector[2] = -LF; //position of end-effector wrt last joint (see Darwin kinematics figure from ASME paper)
            for (i=0;i<3;i++)
               pos_end_effector[i] += body_to_joint[i]; //position of end-effector wrt to com, as com = 0,0,0
            sdpos(ii,pos_end_effector,pos2);
         }
         else if (ii==HEAD_PITCH)
         {
            pos_end_effector[0] = 0; pos_end_effector[1] = 0; //position of end-effector wrt last joint (see Darwin kinematics figure from ASME paper)
            if (HEAD_HEIGHT == 1)
               pos_end_effector[2] = Lh/2+HY;
            else
               pos_end_effector[2] = HY;
            for (i=0;i<3;i++)
               pos_end_effector[i] += body_to_joint[i]; //position of end-effector wrt to com, as com = 0,0,0
            sdpos(ii,pos_end_effector,pos2);
         }
         else
         {
            //This loop tries to find inboard body joint number attached
            for (jj=1;jj<NBOD;jj++)
            {
               sdjnt(jj,info,slider);
               inb_body_no = info[2]; //inboard body number = inboard joint number
               outb_body_no = info[3]; //outboard body number = outboard joint number
               if (inb_body_no==ii)
               {
                  sdgetitj(outb_body_no,inb_to_joint);
                  sdpos(inb_body_no,inb_to_joint,pos2);
               }
            }
         }

         fprintf(fid,"%f %f %f ",pos1[0],pos1[1],pos1[2]);
         fprintf(fid,"%f %f %f ",pos2[0],pos2[1],pos2[2]);
         //count=count+6;


         printf("%f %f %f ",pos1[0],pos1[1],pos1[2]);
```

```
        printf("%f %f %f ",pos2[0],pos2[1],pos2[2]);
        printf("\n");
        //

        //            fprintf(fid,"\n");

    }
    return 0;
}
```

A.2 animation.c code

```
/**********************************************************************************/
/*
  animate.c: animate a data file.
*/

/**********************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "params.h"

#include "drawstuff.h" /* ODE Graphics stuff */
#include "drawstuff-cga.h" /* CGA stuff to make things clearer */
#include "sdlib.h"
#include "le_ludecomp.c"
#include "useful.c"
//#include "params.h"

#define NPOINTS 10000

// ************ Change this as per the problem being solved *********//
#define DATA_PTS 126 //Set this based on columns in data file
float data[NPOINTS][DATA_PTS]; //Structure that will store the data.
char DATA_FILE[] = "data.txt"; //Data file to read
float STEPS;
int data_points;
//int  COUNTER;

double trans_zz = 0.37;

// ********************************************************************** //


static void read_data()
{
    int i, j;
    FILE *fid;
   // int data_points;
```

```
      fid = fopen(DATA_FILE,"r");

//    double dc[3][3];
//   sdang2dc(0,0,0,dc);
//    /* Read file */
//    for (i=0;i<NU;i++)
//    {
//                fscanf(fid,"%f", &link_lengths[i]);
//                printf("%d %f \n",i,link_lengths[i]);
//    }


   i = 0;
//   fscanf(fid, "%d", &data_points);
//   if (data_points>DATA_POINTS);
//   {
//      printf("Please increase DATA_POINTS  \n");
//      exit(1);
//   }

   while(!feof(fid))
   {
      for (j=0;j<DATA_PTS;j++)
         fscanf(fid, "%f", &data[i][j]);

                //printf("%d \n",i);
      i = i+1;

                if (i>NPOINTS)
                {
                        printf("There more simulation steps then alloted memory for data \n");
                        printf("Increase NPOINTS  \n");
                        exit(1);
                }
   }
   fclose(fid);
   STEPS = i-1;



//    /* For Display only */
//   for (i=0;i<1;i++)
//   {
//      for (j=0;j<DATA_PTS;j++)
//         printf("%1.2f ",data[i][j]);
//      printf("\n");
//   }
}



/*************************************************************************/

static void start()
{
```

```
  // set up view point

    //Side view
//     static float xyz[3] = {1.6759,-0.3451,1.0800};
//     static float hpr[3] = {170.5000,-11.0000,0.0000};
//
    //Top view
//    static float xyz[3] = {0.3400,-0.5899,2.2900};
//     static float hpr[3] = {152.5000,-67.0000,0.0000};

    //right side view
//    static float xyz[3] = { 1.3294,0.1563,0.5600};
//     static float hpr[3] = {-173.5000,-0.5000,0.0000};

    //front close up view
    static float xyz[3] = { 0.5725,0.0056,0.3900};
    static float hpr[3] = {-178.5000,2.0000,0.0000};




      dsSetViewpoint (xyz,hpr);
}

/*************************************************************************/
// called when a key pressed

static void command (int cmd)
{
  // don't handle user input yet.
  dsPrint ("received command %d (`%c')\n",cmd,cmd);
}

/*************************************************************************/




////////////////////////////////////////////////////f
static void display (int pause)
{
  //float center[3];
    float RR[12]={0};
       float R[12] = {0};
       float pos1[3], pos2[3], mid_pos[3];
       double dpos[3];
       int ii,i,j;
         //float sides[3] = { WIDTH, HEIGHT, LENGTH };

       static int COUNTER = 0;

       //printf("%d \n",COUNTER);

       j = 0;

       /* NOTE: The R matrix is as follows
```

```
        R = [ R[0] R[1] R[2]  R[3];
        R[4] R[5] R[6]  R[7];
        R[8] R[9] R[10] R[11]];
        The elements R[3], R[7] and R[11] are not used */
       RR[0] = RR[5] = RR[10] = 1;

    //int bodies = data_points/6;
      for (ii=0;ii<NBOD;ii++)
      {

//              center[XX] = data[COUNTER][j]+trans_xx; j = j+1;
//              center[YY] = data[COUNTER][j]+trans_yy; j = j+1;
//              center[ZZ] = data[COUNTER][j]+trans_zz; j = j+1;

//              for ( i = 0; i < 12; i++ )
//                      R[i] = 0;

//              R[0] = data[COUNTER][j];  j = j+1;
//              R[1] = data[COUNTER][j];  j = j+1;
//              R[2] = data[COUNTER][j];  j = j+1;
//              R[4] = data[COUNTER][j];  j = j+1;
//              R[5] = data[COUNTER][j];  j = j+1;
//              R[6] = data[COUNTER][j];  j = j+1;
//              R[8] = data[COUNTER][j];  j = j+1;
//              R[9] = data[COUNTER][j]; j = j+1;
//              R[10] = data[COUNTER][j]; j = j+1;



              pos1[XX] = data[COUNTER][j]; j = j+1;
              pos1[YY] = data[COUNTER][j]; j = j+1;
              pos1[ZZ] = data[COUNTER][j]+ trans_zz; j = j+1;

              pos2[XX] = data[COUNTER][j]; j = j+1;
              pos2[YY] = data[COUNTER][j]; j = j+1;
              pos2[ZZ] = data[COUNTER][j]+ trans_zz; j = j+1;



              float link_length = sqrt((pos2[0]-pos1[0])*(pos2[0]-pos1[0])+
                                       (pos2[1]-pos1[1])*(pos2[1]-pos1[1])+
                                       (pos2[2]-pos1[2])*(pos2[2]-pos1[2]));

              if (link_length!=0)
              {
                      dpos[0] = (pos2[0]-pos1[0])/link_length;
                      dpos[1] = (pos2[1]-pos1[1])/link_length;
                      dpos[2] = (pos2[2]-pos1[2])/link_length;
              }


              //A: Get position of mid-point
              //sdvadd(pos1,pos2,mid_pos); sdvmul(0.5,mid_pos,mid_pos);
              for (i=0;i<3;i++)
                      mid_pos[i] = 0.5*(pos1[i]+pos2[i]);
```

```
		//B: Get orientation matrix

// A method to find R matrix between vector along z axis and line joint 2 joints (Rodriguez formulae)
		double zvec[3]={0,0,1}; //already normalized

//		% Get the axis and angle
//		1) angle = acos(v1'*v2);
	float value = dpos[0]*zvec[0]+dpos[1]*zvec[1]+dpos[2]*zvec[2];
	if (value>=1)
	   value = 1;
	if (value<=-1)
	   value = -1;
	//if (ii==11)
	//    printf("%f \n",temp);
		//float theta = acos(dpos[0]*zvec[0]+dpos[1]*zvec[1]+dpos[2]*zvec[2]);
	float theta = acos(value);//angle between zvector and dpos
	//theta = 0.5;
	//double theta_temp = acos(dpos[0]*zvec[0]+dpos[1]*zvec[1]+dpos[2]*zvec[2]);
	//if (ii==11)
	//    printf("%f \n",theta_temp);

//		OR theta = acos((pos2[ZZ]-pos1[ZZ])/length);

//		2) axis = cross(v1,v2)/norm(cross(v1,v2));
		double axis[3];
		sdvcross(zvec,dpos,axis);
		double l_axis =   sqrt(axis[0]*axis[0]+axis[1]*axis[1]+axis[2]*axis[2]);
	//if (ii==11)
	//    printf("%f \n",l_axis);
		if (l_axis!=0) //normalize if length is not equal to zero
		{
			axis[0] = axis[0]/l_axis;
			axis[1] = axis[1]/l_axis;
			axis[2] = axis[2]/l_axis;
		}

//		 % A skew symmetric representation of the normalized axis
//		 3) axis_skewed = [ 0 -axis(3) axis(2) ; axis(3) 0 -axis(1) ; -axis(2) axis(1) 0];
		double axis_skewed[3][3]={0};
		axis_skewed[0][1] = -axis[2];
		axis_skewed[0][2] = axis[1];
		axis_skewed[1][0] = axis[2];
		axis_skewed[1][2] = -axis[0];
		axis_skewed[2][0] = -axis[1];
		axis_skewed[2][1] = axis[0];
//
//		 % Rodrigues formula for the rotation matrix
//		 4) R = eye(3) + sin(angle)*axis_skewed + (1-cos(angle))*axis_skewed*axis_skewed;
	double I[3][3], r[3][3]={0};
	double A[3][3], B[3][3]={0};

		identity(3, &I[0][0]);
		multiplySCALAR2MAT(3, 3, &axis_skewed[0][0], sin(theta), &A[0][0]);
		multiplyMAT2MAT(3, 3, 3, 3, &axis_skewed[0][0], &axis_skewed[0][0], &B[0][0]);

//	 int iii,jjj;
```

```
//      if (ii==11)
//      {
//         for(iii=0;iii<3;iii++)
//         {
//            for(jjj=0;jjj<3;jjj++)
//            {
//               printf("%f ",B[iii][jjj]);
//            }
//            printf("\n");
//         }
//         printf("\n");
//      }


      float one_cos = (1-cos(theta));
      //if (ii==11)
      //   printf("%f \n",theta);
            multiplySCALAR2MAT(3, 3, &B[0][0], one_cos , &B[0][0]);



            add(3, 3, &I[0][0], &A[0][0], &r[0][0]);

//      int iii,jjj;
//      if (ii==11)
//      {
//         for(iii=0;iii<3;iii++)
//         {
//            for(jjj=0;jjj<3;jjj++)
//            {
//               printf("%f ",B[iii][jjj]);
//            }
//            printf("\n");
//         }
//         printf("\n");
//      }

            add(3, 3, &B[0][0], &r[0][0], &r[0][0]);
//      int iii,jjj;
//      if (ii==11)
//      {
//      for(iii=0;iii<3;iii++)
//      {
//         for(jjj=0;jjj<3;jjj++)
//         {
//            printf("%f ",B[iii][jjj]);
//         }
//         printf("\n");
//      }
//      printf("\n");
//      }



            //5) Convert r matrix to something which draw-stuff understands
            for ( i = 0; i < 12; i++ )
```

```
{
                        R[i] = 0;
   //if (ii==11)
     //  printf("%f ",R[i]);
}
//if (ii==11)
//printf("\n");


            for ( i = 0; i < 3; i++ )
            {
                        R[i] = r[0][i];
                        R[i+4] = r[1][i];
                        R[i+8] = r[2][i];

            }
/*if (ii==11)
{
   for (i=0;i<12;i++)
   printf("%f ",R[i]);

}
            printf("\n");*/

            //Now draw something
//for (i=0;i<12;i++)
//    printf("%f ,", R[i]);
//    printf("\n");

            //Put spheres at the joints
            dsSetTexture (DS_WOOD);
            dsSetColor (0,1,0);
            dsDrawSphere(pos2,RR,0.015);

            //Option 1: to use lines
            //                      dsSetTexture (DS_WOOD);
            //                      dsSetColor (1,0,0);
            //                      dsDrawLine(pos1,pos2);

            //Option 2: Use solids (box or capsule)
            dsSetTexture (DS_WOOD);
            //dsSetColor (0,0,0); //black
dsSetColor (1,1,1); //gray
//ooiijjj
//printf("%f %f",robot_dim[ii][0],robot_dim[ii][1]);

            float dimensions[3] = {robot_dim[ii][0],robot_dim[ii][1],link_length};
//if(ii==11 || ii==12)
//{
   dsDrawBox(mid_pos,R,dimensions);
   //printf("%f %f %f \n", mid_pos[0],mid_pos[1],mid_pos[2]);
   //for (i=0;i<12;i++)
      // printf("%f ,", R[i]);
   //printf("\n");
            //dsDrawCapsule(mid_pos,R,link_length,0.01);
//dsDrawCylinder(mid_pos,R,link_length,0.01);
```

```
    //}
    }



    // Various geometries that can be animated //
//  switch( STR )
//  {
//          case 'b':
//                  /* Draw a box */
//                  dsSetTexture (DS_WOOD);
//                  dsSetColor (0,0,1);
//                  dsDrawBox( center, R, sides );
//                  break;
//          case 's':
//                  /* Draw a sphere */
//                  dsSetTexture (DS_WOOD);
//                  dsSetColor (0,1,0);
//                  dsDrawSphere(center,R,RADIUS);
//                  break;
//          case 'c':
//                  /* Draw a cylinder */
//                  dsSetTexture (DS_WOOD);
//                  dsSetColor (1,0,0);
//                  dsDrawCylinder(center,R,LENGTH,RADIUS);
//                  break;
//          case 'p':
//                  /* Draw a capsule */
//                  dsSetTexture (DS_WOOD);
//                  dsSetColor (0.5,0.5,0.5);
//                  dsDrawCapsule(center,R,LENGTH,RADIUS);
//                  break;
//          default :
//                  /* Draw a box */
//                  dsSetTexture (DS_WOOD);
//                  dsSetColor (0,0,1);
//                  dsDrawBox( center, R, sides );
//  }


    COUNTER+=1;

    if(COUNTER>=STEPS)
            COUNTER = 0; //reset counter


    /* Delay */
  //double f;
    //for( f = 0.0; f < 2000000.0; f += 0.1 );



}
```

```
/************************************************************************/
/************************************************************************/

//void
//sduforce(double t, double *q, double *u)
//{}

int main (int argc, char **argv)
{

      // Read data and load this into memory
      read_data();

//    printf("/*********************************/");
/*    printf("\nWhich geometry do you want to be animated? \n");
      printf("b = box \ns = sphere \nc = cylinder \np = capsule\n");
      printf("Enter a choice \n");
      scanf("%c",&STR);*/

   dsFunctions fn;

  // setup pointers to drawstuff callback functions
  fn.version = DS_VERSION;
  fn.start = &start;
  fn.step = &display;
  fn.command = &command;
  fn.stop = 0;
#ifdef WIN32
  fn.path_to_textures = "C:/cga/kdc/sim5/useful/drawstuff-windows/textures";
#else
  //Change to this appropriately
  //              fn.path_to_textures              =              "/Users/pab47/Documents/DISK/template_files/template_C/animation-
drawstuff/home_comp/useful/drawstuff-linux/textures";
     fn.path_to_textures = "/Users/andrewwaterreus/Desktop/darwin_animation/drawstuff/textures";
//      fn.path_to_textures   =   "/Users/pranavb/Documents/pranavb-macbook/2014Backup/template_files/template_C/animation-
drawstuff/all_geometries/drawstuff/textures";
  //                    fn.path_to_textures       =       "/Users/pranavb/Documents/2014Backup/template_files/template_C/animation-
drawstuff/all_geometries/drawstuff/textures";
#endif


  // do display
  dsSimulationLoop( argc, argv, /* command line arguments */
              2*352, 2*288, /* window size */
               &fn ); /* callback info */


  return 0;
}

/************************************************************************/
```

B. Images

B.1 DARWIN OP2 Robot

B.2 VICON motion capture system