Kurt Lee
5/5/2015
CIS II: Group 12

Final Report: The Robotic Ear Nose and Throat Microsurgery System (REMS): IRB Study, Calibration, and Tool Rotation Tracking

1 Introduction

1.1 Background and Relevance

The Robotic Ear Nose and Throat Microsurgery System (REMS) is a robot developed by our mentor Kevin Olds in the JHU Laboratory for Computational Sensing + Robotics (LFCSR). The REMS project hopes to address the challenges that face conventional, nonrobotic approaches to endoscopic surgery of the paranasal sinuses. By holding the tool with the surgeon, the REMS mitigates any unwanted motion like the tremor in the surgeon's hand. The REMS also provides internal tracking, which is essential because workspace visualization in most sinus surgeries is not trivial. Combined, these features allow the REMS to ensure that critical anatomy is almost never unintentionally touched or damaged (the robot will even alert the surgeon when they get near "danger zones" by emitting beeps).

1.2 Project Review

Our project for the [600.446] Advanced Computer Integrated Surgery (CIS II) course actually consists of three distinct subprojects, all related to the REMS.

1.2.1 Subproject 1: IRB Study

The first project is an IRB Validation Study and is currently ongoing. The primary goal of this study is to determine if robotic surgical assistance with the REMS improves surgical skill compared to conventional (unaided) surgery. Participants are asked to navigate a tracked instrument to contact specific anatomical targets in the sinuses of a cadaver head. The study involves 20 novice (untrained) participants and 5 expert (professional) surgeons. The 20 novices are randomly divided into two groups of ten. One group is trained conventionally and one group is trained using the robotic method. At the end, the groups are assessed with a standard evaluation method to determine if robotic surgical assistance increases surgical skill.

1.2.2 Subproject 2: Tool Tip Calibration

The second project involves additional calibration of the REMS. Currently, the REMS can achieve sub-millimeter precision of the tooltip location when no force is being applied to the tool. The problem currently is that many parts of the robot (specifically the area around the force sensor) are not perfectly stiff. This means the tooltip deflects by a few millimeters when the tool is pushed hard enough. This deflection is not tracked by the robot since the robot kinematics do not change with this deflection. However, the amount of force being applied is logged. To calibrate for current and future errors, we have designed a camera tracking system and calibration code that records the position of the tool tip independent of the robot kinematics. This position can be compared to where the robot thinks the tool is, and the difference in position can be correlated with the force vectors the robot has recorded.

1.2.3 Subproject 3: Rotation Tracking

The third subproject seeks to add an additional degree of freedom. Currently, the REMS has no way of tracking the rotation of a tool along the tool axis. Any tool that can currently rotate while attached to the REMS rotates freely. While this is not a concern for straight tip tools, this presents a challenge for tooltip tracking of tools with burred or bent tips. Options to fix this were to either create a device with a motor that would add this rotation to the kinematics of the robot or passively track the rotation of a tool. Given the lack of real estate in the tool holder area of the robot, we have opted to go for a passive solution. Our solution involves using an absolute rotary encoder that would be an attachment to existing tools. A small webcam would be mounted on the REMS close to the tool. The camera will be able to read the angle data encoded in the rotary encoder and send this data to the REMS in order to track the exact position of the tooltip.

2 Technical Summary – Project 1: IRB Study

2.1 Overview

The project is a Homewood IRB approved study aiming to determine whether robotic surgical assistance improves surgical skill compared to unaided surgery. The study is at time of writing (5/6/15) currently ongoing due to unavoidable delays we experienced throughout the semester. These included the limited availability our mentor Dr. Ishii, as well as unexpected damaging of the REMS by other parties. We plan to continue the study past the poster presentation for this course: participants will come in through the rest of the finals period and the summer.

2.2 Preparation

Required for the study is a single alcohol fixed human cadaver head with CT marking screws fixed onto the head. The first head did not contain the needed structures so another head was acquired. We affixed new screws to this head after the previous screws proved inadequate in enabling navigation of the sinus area. With a CT scan, we identified the screw locations in CT coordinates via open-source program 3D slicer. With these CT scanned screws, we are able to register the head to the robot by touching the robot tool tip to each of the 6 screws. This enables us to track the tool tip inside the head.

Our mentor Dr. Ishii (an ENT surgeon at Hopkins Bayview) provided 5 expert tool trajectories for touching each of four target structures in the sinuses – these were stored in the robot. These 5 expert tool trajectories for each target structure will be averaged together to provide a smooth path to follow. The recorded trajectories serve as a virtual fixture or “path” that the robot will constrain tool movement along. Due to the nature of this study, additional preparation will involve training one group of 10 novices with the REMS and another group of 10 novices with the traditional freehand method.

2.3 Study Protocol

The task at hand is to contact four distinct structures within the sinuses with the tip of a tracked tool. These are: the middle turbinate attachment point, the superior turbinate, the spheno-ethmoidal junction, and the Eustachian tube.

Each participant is given a brief introduction to sinus anatomy and sinus surgery. The participant receives 30 minutes of training/instruction in the surgical task with either the robotic or freehand method (corresponding to their group of with robot or without robot). They are then given an additional 30 minutes to practice the task on their own. This entails performing the navigation to hit one of the target structures. Even in practice, the subject is required to complete the task once started. All training and practice takes place in the head's left nostril, and the testing occurs in the head's right nostril. Throughout these 60 minutes, a looping video is displayed showing clips of Dr. Ishii's navigation to the target structures in the sinuses. After the training and practice phases, the participant is asked to perform the same task (contacting the same structures) in the other (evaluation) nostril in 30 minutes. They are evaluated freehand, with the robot, and with the robot under the stored path constraints, each 5 times. During the evaluation, tool trajectories and kinematic information will be recorded through the REMS for robotic evaluations, and through an EM tracker for freehand evaluation. Endoscopic video showing tool movement will also be recorded. Participant tracking data will be compared to expert data with ANOVA to determine which group operated with the most surgical skill. We will also investigate whether robotically trained participants retained skill when switching to freehand operation.

2.4 Analysis and Results

Due to the ongoing nature of our study, we have not yet accumulated any significant data results. All of the preparatory work and setup has been accomplished, and all the participants have been recruited. We will bring the participants in throughout the rest of finals period and also over the summer. A "dry" run with a trial participant has also been conducted in preparation for recorded runs. The exact methods of data analysis are not yet finalized, and will be decided over the summer in consultation with Drs. Ishii, Vedula, and Olds.

3 Technical Summary – Project 2: Robot Calibration

3.1 Problem

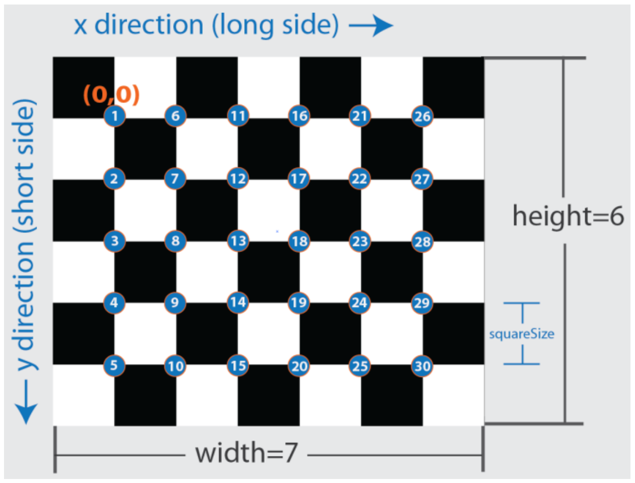
Currently, the REMS can achieve sub-millimeter precision of the tooltip location when no force is being applied to the tool. The problem is that many parts of the robot (specifically the area around the force sensor) are not perfectly stiff. This means the tooltip deflects by a few millimeters when the tool is pushed hard enough. This deflection is not tracked by the robot since the robot kinematics do not change with this deflection. However, the amount of force being applied is logged. Thus it is possible to design a setup that can correlate force data with deflection data. What we hope to achieve is to create a system that can accurately measure deflection.

3.2 Approach and Algorithms

To calibrate for current and future errors, we have designed a camera tracking system and calibration code that records the position of the tool tip independent of the robot kinematics in the coordinate system of the robot. This position can be compared to where the robot thinks the tool is, and the difference in position can be correlated with the force vectors the robot has recorded.

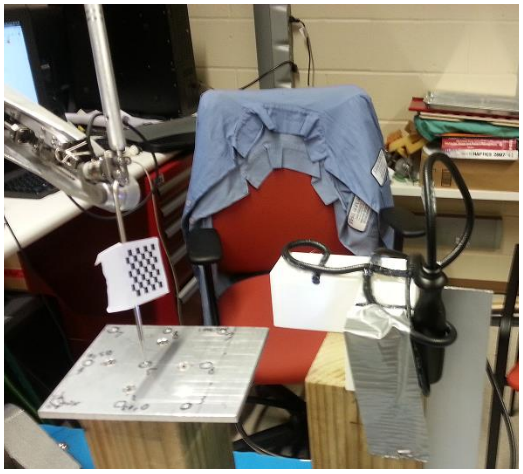
Originally, the design was to use two cameras rigidly attached to a frame at approximately a 90 degree angle. Both these cameras were calibrated using the Caltech Computer Vision Toolbox in order to get their distortion, principal point, and focal length. The two cameras were also stereo calibrated with each other in order to get the world coordinates of right camera in the coordinates. These two cameras would be used to track a segmentable green ball that would be rigidly attached to the tool of the camera. The two cameras would find the 3D coordinate of the centroid of the green ball.

Quickly, we found out this approach would not only lead to high errors but also was incapable of actually locating the tooltip position. This is because getting only one coordinate (the ball centroid) does not encode rotation, only translation from the tip. Additionally, getting the true centroid of the ball proved to be quite difficult as the green ball would rarely segment properly in an image.



Thus, we arrived at our current setup which has so far proven successful. Our current setup requires only one calibrated camera (thus getting rid of the error from a stereo calibration) and a printed out checkerboard pattern with known dimensions (measured with a digital caliper). Using the MATLAB computer vision toolbox, we are able to get the 2D image coordinates of the checkerboard corner points to subpixel accuracy. From this, the extrinsics of the checkerboard (rotation and translation of the origin [top left] on the checkerboard) can be calculated using the intrinsic parameters of camera. All this is calculated using the MATLAB toolbox.

With the rotation and translation of the checkerboard able to be computed, it is thus possible for our camera to not only know the position of the tooltip, but also the position of this tooltip in robot coordinates, based on only an image with the checkerboard in it. This can be done with a two-step calibration-registration process. First, a checkerboard is attached (usually with tape) to the tool of the robot, roughly halfway up on the tool. The camera is placed such that the checkerboard is always in the field of view of the camera and the camera is firmly attached to something that won't move. The entire setup can be seen on the right. We then do a standard pivot calibration of the tooltip, taking 5 images of the checkerboard at various poses of the tool. Once the tooltip position is known, a point cloud to point cloud registration is done between the robot and the camera by moving the tooltip to 6 random positions in space (evenly spaced apart). The position of the tooltip in camera coordinates is known because of the prior pivot calibration and the position of the tooltip in robot coordinates can be extracted from the force/position log of the REMS desktop. A point cloud to point cloud registration is done on these 6 points using Aron's method to calculate the frame transformation from camera coordinates to robot coordinates. Once this two-step calibration is done, the true pointer tip location can be known in robot coordinates simply with our camera seeing an image of the checkerboard, which can be considered an impressive feat if this could be done to sub-millimeter accuracy.



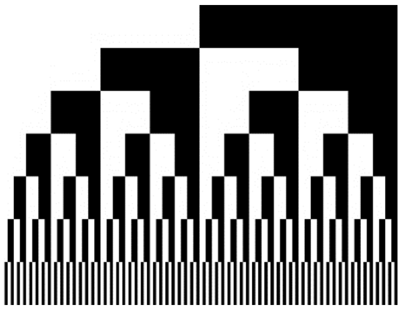
4 Technical Summary – Project 3: Tool Rotation Tracking

4.1 Problem

The REMS currently cannot track tool rotation about the tool axis. Any tool that currently can rotate while attached to the REMS does so freely. This is an issue when the tooltip is burred or bent because the REMS cannot accurately track the position of the tip without knowing the angular rotation of the tool.

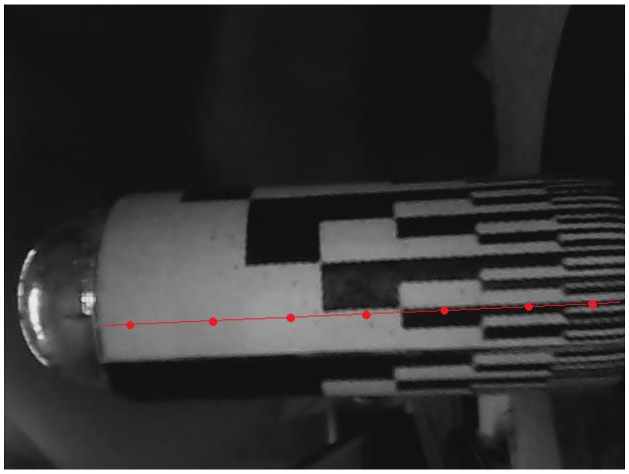
4.2 Approach and Algorithms

To address this problem, our team has created a prototype rotary encoder system designed to be an attachable and detachable module on the REMS. The setup includes an absolute rotary encoder pattern that can be attached to the cylindrical shaft of a tool. The pattern, shown right, encodes 128 different values. This means the encoder pattern has a resolution of 2.8125 (360/128) degrees. Specifically, the number of different values equals 2^N where N is the number of layers. We decided 7 layers was a good value with which to create a prototype. While 8 layers would give close to single degree precision, the printers on campus simply do not have a high enough DPI to accurately print the 8th layer (258 alternating black and white stripes).



To track the rotary encoder pattern, we bought a generic 2 megapixel lipstick webcam from Amazon. The encoder pattern was carefully glued to a rotating cylinder approximately 2 cm in diameter (roughly the diameter of a surgical tool). Both the camera and cylinder were then rigidly attached to a piece of plexi-glass so that the entire pattern could be seen in the field of view of the camera. This setup would simulate an eventual implementation on the REMS, where the camera would be attached on the last arm of the robot (so the cylinder itself doesn't move translationally with respect to the camera).

On the software side, the setup is run in MATLAB and is fairly straightforward although many iterations of detection methods were used until one was settled upon to be the best. The initial method took a straight line along the axis of the cylinder (as seen on the right), and sampled 7 pixels evenly spread out along the line. The grayscale pixel values were thresholded for black and white. This outputs a 7 value row vector of 0's or 1's (white=0 black=1). The actual angle value of this row vector is simply (360/128)*X where is X is the row vector (which is actually just a number in base 2) converted into a base 10 number.



Since our camera generates its own light source, there was no trouble separating black from white. Despite this, the initial method generated insanely high error rates because individual pixels did not always correspond to the actual color of the encoder block. This is just due to imperfections in the image itself as well as the printing process of the encoder pattern. To correct for this, instead of taking just a single pixel value along this line, we sampled a circle of pixels at each of the same points. Each circle has a diameter of 7 pixels (this is the width of a block on the 7th layer of the encoder pattern). The value of the circle (white or black) is determined by which color takes up >50% of the pixels within each circle. This proved to be a much more robust method of detection which can be seen in the results section below. As a note, the current setup uses a sampling of rate of 1 Hz, which we acknowledge is quite slow. Of course, this could be raised significantly by lowering the refresh timer as well as getting a higher fps camera.

4.3 Results and Significance

In order to test the accuracy of our detection algorithm, 50 sample images of the cylinder randomly being rotated were taken from the camera and correlated with the output angle associated with each image. These images were examined manually to see if the output from the computer matched the true angle associated with the image. The detection algorithm ended up with a 94±3.3% correct output rate. We believe this error is mainly associated with the transition between encoding blocks. It is likely one layer's block may be detected as changing before another layer's. This would lead to an incorrect output of angle. This transitional error seems to happen more often at some transitions more than others, which leads us to believe this error is due to the poor print quality of the pattern, and a higher quality print would remove this residual error.

The significance of this successful prototype implementation of an optical rotary encoder is that this could be easily attached to the REMS. The footprint of a webcam is exceptionally small and the camera output could be sent directly to the REMS desktop. An encoder pattern can easily be glued onto tools as well. Overall, we believe using an optical rotary encoder to track tool rotation is a simple, effective, and cost efficient method to improve the REMS.

Kurt Lee
5/5/2015
CIS II: Group 12

5 Management Summary

5.1 Member Responsibilities

Given the sprawling nature of our project and what we hoped to accomplish, effective management was vital. We decided that each member would be in charge of their own subproject.

- Kurt: IRB Study (Subproject 1)
- Brian: Tool Tip Calibration (Subproject 2)
- Barbara: Rotation Tracking (Subproject 3)

While each member was heavily involved with all aspects of the entire project, the leader of each subproject was responsible for planning and progress tracking within that subproject.

5.2 What We Accomplished

5.2.1 Subproject 1: IRB Study

We initially hoped to complete the entire study within the semester: collection of data for all 20 participants as well as results, analysis, and interpretation. Unfortunately, we encountered many difficulties which we will enumerate here. A large dependency was the availability of Dr. Ishii, which was extremely difficult to fulfill. Compounded on that, set up of the robot took several weeks because we had to get information, setup code, and obtain further assistance from various people such as Marcin, Kevin, Swaroop, and Narges. In addition, the first head we obtained was not adequate because it did not contain the relevant anatomical structures in the nose. The second head we obtained contained the relevant structures but was filled with fluid and its screws were placed too far apart for comfortable navigation of the sinus area. This required extra time to get rid of the fluid and to place new screws in the head and to get a new CT scan, as well. Finally, the machine's force sensor broke in the last few weeks, rendering it unusable. Due to the unavoidable setbacks described (machine malfunction and breakage by other group, scheduling issues with Dr. Ishii, etc.), we were not able to record all 20 participants before the date of the poster presentation. However, we successfully prepared the experimental setup and also conducted a successful trial run. In consultation with our mentor Dr. Ishii, we have planned to conduct the remainder of the study over the next couple of weeks and also the summer.

5.2.2 Subproject 2: Tool Tip Calibration

The original goal of subproject 2 was to actually have a calibration model for the deflection error done for the REMS. This could have actually been done feasibly in the time we had left. However, considering the force sensor broke on the last week of the semester (and this is the area where most of the deflection occurs), we decided it was not worth doing calibration on the current REMS, since this data would be rendered mostly incorrect when the REMS is repaired.

Noting this, we have still created the setup and software for any future calibration to be done. Our camera was calibrated using 40 calibration images and the Caltech Computer Vision Toolbox. Code was written in Python and MATLAB to find the true position of the pointer tip in robot coordinates, smoothly acquire timestamped images from the camera (these images correspond to the true coordinates of the pointer tip at any time), match these images with the corresponding position/force values from the robot and output this in a .csv. This process is all done automatically by our MATLAB script as long as the images and the robot position/force log are put in the same folder.

5.2.3 Subproject 3: Rotational Tracking

We accomplished our goals for subproject 3. Our original goal for subproject 3 was to create a proof of concept prototype of a device that could attach to the REMS and give it a 6th DOF (rotation of the tool about its tool axis). Originally, it was conceived that we would probably create some sort of CAD model of a device with a rotor that would actively rotate the tool. As we brainstormed about this problem, we decided some sort of passive rotation tracking implementation would be the most ergonomic solution because the amount of space near this part of the robot is severely limited. One of the constraints of the REMS is that the area around the tool must be as free as possible for the surgeon. Because our group already had considerable knowledge working with cameras from subproject 2, the idea to use an optical rotary encoder was used.

We created a prototype encoder using a cheap webcam and a poor DPI printout of the encoder pattern and showed that even this rudimentary implementation works very well. Because of this, as well as the continued blessing of our mentor Kevin Olds, we believe this to be a viable solution to the problem and will continue working on it.

5.3 Looking Forward

Over the next few months, we will continue to be involved with the REMS with Drs. Ishii, Vedula and Olds. This most notably involves conducting the IRB study with the recruited novices, and interpreting the results. Through the study, we hope to learn about and demonstrate the efficacy of the REMS in improving surgical skill compared to conventional training.

Once the REMS has its original force sensor reinstalled, we plan to carry out the deflection calibration using the setup created. Currently, the REMS is operating in a bootleg fashion. The force sensor installed is not the same model and some bolts are intentionally not screwed in tightly. This means calibration of the current setup would be meaningless.

Finally, we plan to continue working on both the physical implementation as well as the software side of our tool rotation tracking prototype. The next step would be to create more robust code that will compensate for jitter in the tool as well as give an alarm if the rotary encoder is occluded by an object or hand. Additionally, we would create our own prototype tool with the encoder pattern cleanly printed on so testing on the actual REMS can be done.

6 Technical Appendices

Subproject 2 Code

Stereo.m- contains intrinsic camera parameters for the cameras we are using

```
unmarkintrinsics=[802.896445869757140 0 0;0 807.382050089614840
0;299.954971177534730  229.967083874900480 1];
unmarkradial=[0.283142557578893  -1.217241727737446 0];
unmarktangential=[-0.005308257509087 ; -0.002244189383837];
% markintrinsics=[819.041452444198060 0 0;0 818.835334482124610
0;303.286025334390900 247.225438197867190 1];
% markradial=[0.286167182391264 -1.575367170935510 0];
% marktangential=[-0.004942282627839 ; 0.002757042848404];
unmarkCamParam=cameraParameters('IntrinsicMatrix',unmarkintrinsics,'RadialDis
tortion',unmarkradial,'TangentialDistortion',unmarktangential);
%
markCamParam=cameraParameters('IntrinsicMatrix',markintrinsics,'RadialDistort
ion',markradial,'TangentialDistortion',marktangential);
% unmarktomarkRot=[0.1881    -0.6942    0.6948
%    -0.1480    0.6793    0.7188
%    -0.9709   -0.2380    0.0250];
% unmarktomarkTran= [ -27.4715
%   -30.1886
%    41.7396];
%
stereoParams=stereoParameters(markCamParam,unmarkCamParam,unmarktomarkRot,unm
arktomarkTran);
% unmark1=[99 149];
% mark1=[356 238];
```

Findtip.m – Takes 5 images from a pivot calibration to find Pcal and Ptip (CIS I terminology)

```
stereo;
[worldPoints] = generateCheckerboardPoints([7, 8],5.32);
bigmat=zeros(3*5,6);
for i=1:5
    bigmat((3*(i-1)+1):(3*(i-1)+3),(4:6))=-(eye(3));
end
transmatrix=zeros(3*5,1);
imagefiles = dir('*.png');
for ii=1:5
   currentfilename = imagefiles(ii).name;
   currentimage = imread(currentfilename);
   currentimage = undistortImage(currentimage, unmarkCamParam);
   gay=currentimage(10:470,10:630,:);
   [imagePoints,boardSize] = detectCheckerboardPoints(gay);
   imagePoints=imagePoints+9;
   [rotationMatrix, translationVector] = extrinsics(imagePoints, worldPoints,
unmarkCamParam);
   rotationMatrix=rotationMatrix';
   translationVector1=translationVector';
   bigmat((3*(ii-1)+1):(3*(ii-1)+3),(1:3))=rotationMatrix;
   transmatrix((3*(ii-1)+1):(3*(ii-1)+3),1)=translationVector1;
end
pcalppiv=bigmat\(-transmatrix);
ppiv=pcalppiv(4:6);
pcal=pcalppiv(1:3);%pcal is pointer in board coordinates ppiv is pointer in
cam coordinates
errorpls=zeros(3,5);
for i=1:5
```

```
    errorpls(1:3,i)=abs((bigmat((3*(i-1)+1):(3*(i-
1)+3),(1:3))*pcal)+transmatrix((3*(i-1)+1):(3*(i-1)+3),1)-ppiv);
end
meanerror=[mean(errorpls(1,:));mean(errorpls(2,:));mean(errorpls(3,:))];
stderror=[std(errorpls(1,:));std(errorpls(2,:));std(errorpls(3,:))];
```

camtorob.m- takes 6 images from camera, finds the pointer tip location in camera coordinates using output from findtip.m, finds corresponding pointer tip locations in robot coordinates by parsing robot data log, performs rigid body frame transformation to find rotation and translation to take camera coordinates to robot coordinates

```
alldata = csvread('cloud2.csv',1,0);
alldata(:,1)=round(alldata(:,1),1);
pointertips=zeros(12,3);
imagefiles = dir('*.png');
for ii=6:11
   currentfilename = imagefiles(ii).name;
   currentimage = imread(currentfilename);
   currentimage = undistortImage(currentimage, unmarkCamParam);
   gay=rgb2gray(currentimage(10:470,10:630,:));
   [imagePoints,boardSize] = detectCheckerboardPoints(gay);
   imagePoints=imagePoints+9;
   [rotationMatrix, translationVector] = extrinsics(imagePoints, worldPoints,
unmarkCamParam);
   rotationMatrix=rotationMatrix';
   translationVector1=translationVector';
   ppiv=rotationMatrix*pcal+translationVector1;
   [pathstr,name,ext] = fileparts(currentfilename);
   time=round(str2double(name),1);
   [row, col]=find(alldata==time,1);
   pointertips((ii-5),:)=(ppiv)';
   pointertips((ii+1),:)=alldata(row,((col)+1):((col)+3));

end
[rotrob, transrob] = FrameTransformation(pointertips, 6, 1);%outputs
transformation to take camera coordinates to robot coordinates
```

Finalproj2.m-Takes all remaining images in the folder and finds the location of the pointer tip in robot coordinates. Parses the robot position/force log to find the closest timestamped values and outputs the true location of the tip vs where the robot thinks the tip is along with the corresponding force being applied on the robot at that timestamp. Outputs all the data as a csv.

```
% Get list of all BMP files in this directory
% DIR returns as a structure array.  You will need to use () and . to get
% the file names.
%first 5 images are for pivot calibration: find pointer tip in camera
%coordinates
%next 6 images are for point cloud to point cloud: camera coordinates to
%robot coordinates
```

```
%alldata = csvread('cloud3.csv',1,0);
%alldata(:,1)=round(alldata(:,1),1);
%[worldPoints] = generateCheckerboardPoints([6, 9],3.27);
%stereo;
%findtip;
%camtorob;
imagefiles = dir('*.png');
nfiles = length(imagefiles);    % Number of files found
rawdata=[];
for ii=12:nfiles
   currentfilename = imagefiles(ii).name;
   currentimage = imread(currentfilename);
   currentimage = undistortImage(currentimage, unmarkCamParam);
   gay=currentimage(10:470,10:630,:);
   [imagePoints,boardSize] = detectCheckerboardPoints(currentimage);
   imagePoints=imagePoints+9;
   [rotationMatrix, translationVector] = extrinsics(imagePoints, worldPoints, 
unmarkCamParam);
   rotationMatrix=rotationMatrix';
   translationVector1=translationVector';
   ppiv=rotationMatrix*pcal+translationVector1;
   [pathstr,name,ext] = fileparts(currentfilename);
   time=str2double(name);
   [row, col]=find(alldata==time,1);
   tiprob=rotrob*ppiv+transrob; %tip in robot coordinates
   rawdata=[rawdata;alldata(row,:) tiprob'];

end
csvwrite('calibration',rawdata);
```

Subproject 3 code

Generatesamplepoints2.m- locates the points on the image plane that need to be sampled in order to get encoder value

```
xf=103;
yf=331;
xl=634;
yl=308;
t=[0:1/14:1];
x=zeros(1,7);
y=zeros(1,7);
xy=zeros(480,640,7);
count=0;
for i=[2 4 6 8 10 12 14]
count=count+1;
x(count)=floor(xf*(1-t(i))+xl*t(i));
y(count)=floor(yf*(1-t(i))+yl*t(i));
end
for k=1:7
    for i=1:480
        for j=1:640
            if sqrt(((j-x(k))^2)+((i-y(k))^2))<=3
                xy(i,j,k)=1;
            end
        end
    end
end
```

segmenter2.m-takes the current image from the webcam and outputs the encoded value seen on the cylinder using the image points from generatesamplepoints.m

```
function [ encoded ] = segmenter2(leinput, xy)
theinput=im2bw(leinput,.25);
encoded=zeros(1,7);
for i=1:7
[row, col]=find(xy(:,:,i));
total=0;
for j=1:length(row)
    total=total+theinput(row(j),col(j));
end
total=round(total/length(row));
encoded(i)=total;
end
end
```

angle.m- calculates the angle using the encoded value output from segmenter2.m

```
function [outputangle]=angle(encode) %encode is a 7 entry column vector 0
entry= white 1 entry=black
number=(64*(1-encode(1)))+(32*(1-encode(2)))+(16*(1-encode(3)))+(8*(1-
encode(4)))+(4*(1-encode(5)))+(2*(1-encode(6)))+(1*(1-encode(7)));
outputangle=(360/128)*number;
end
```

rotencode.m- runs the above 3 functions to output the current angle of the cylinder every 1 second

```
vid = videoinput('winvideo', 1, 'YUY2_640x480');
src = getselectedsource(vid);
vid.ReturnedColorspace = 'grayscale';
generatesamplepoints2;
p=1;
saved=[];
savedframes=zeros(480,640,50);
thecount=1;
while p==1
frame = getsnapshot(vid);
encoder=segmenter2(frame,xy);
% frame(327,103)=255;
% frame(308,634)=255;
% savedframes(:,:,thecount)=frame;
% thecount=thecount+1;
current_angle=angle(encoder)
saved=[saved;current_angle];
pause(1);
end
```

Kurt Lee
5/5/2015
CIS II: Group 12