THE ART OF MANIPULATION

Erica Baller
April 30, 2006
CPSC 490b Senior Project
Advisor: Brian Scassellati

I. Abstract

In the Yale Social Robotics Laboratory, a common goal is to further the study of what qualities are perceived to represent human interaction and to integrate this research in a hardware robot in a way that the behaviors and actions of a robot represent that of a human. The primary difference between humans and robots is that humans are thought to be autonomous—indepen-dent in mind and thought with self-directed actions—while robots, simply by their nature of being built and programmed or monitored by humans, are not. In Social Robotics is only by establishing what behaviors represent social autonomy and being able to accurately represent them in a robot system can that system be considered truly “social.”

The word “social” can be simply defined as the interaction of the individual to members of society. It is derived from the Latin *socius*: ally, companion, associate, man. Therefore, a social robot must not only act like a human in behavior but demonstrate characteristics found in a companionship. This implies that communication is essential in distinguishing between a social and non-social entity and in order for a social robot to be developed; it must be able to exhibit communication.

In this lab, *The Art of Manipulation*, communication between humans and robots is explored by evaluating a robot’s ability to successfully manipulate a human being to do a complicated, arbitrary action using minimal motor feedback. It aims to investigate how much and which kinds of motor behaviors generate the best feedback from the subject and to use that data to develop a more specific set of behaviors that demonstrate high accuracy in their ability to persuade and therefore communicate. The general program begins with a motor command and then engages a feedback loop involving analyzing the person’s behavior with respect to the desired goal and giving appropriate positive or negative motor feedback based on the behavior.

II. Background

II.A - Infant Cognition and Communication

Cognitive research of language development has demonstrated that infants seem to have an innate capacity to understand language. Studies have shown that from as early as 6 to 9 months of age, infants have a functioning perceptual information-processing system with respect to cognition and language development. They demonstrate understanding of basic information but do not have the linguistic skills or development to articulate their cognition (4). Based on this research, a “language” called Baby Sign is being developed, studied and implemented for both hearing-impaired and children with normal auditory functioning.1 Many methods for teaching Baby Sign exist but all are variations of a process in which the teacher (usually a parent) conditions their baby to

---

1 Currently, there is no uniform Baby Sign Language Dictionary; different dictionaries vary in number of signs but are fairly consistent across movements for equivalent signs.
associate certain gestures with certain actions, desires, representations, or statuses. Many babies have learned this limited form of communication, some as early as 9 months old. Through these simple signs, babies can communicate and manipulate the actions of their caretakers without words (examples of Baby Sign dictionaries can be found in Appendix A) (7).

II.B - Baby Sign and Social Robotics

The ability of a human being to actively manipulate an adult to do complex behaviors from very simple actions suggests that communication requires a low level of motion and perception. If that were true, it might be possible to find an upper-bound on the necessary complexity of a behavior or how many and which behaviors one must exhibit in order to actively and successfully communicate without words. Provided how central good communication is to creating a true social robot, an analysis of peoples’ success in their responses to different actions could be integrated into a robot to further its progression toward being totally “social.”

III. Problem

One of the main projects in the Yale Social Robotics Lab surrounds the robot Nico. Nico is a one-armed robot with a head and a torso. It has the size, proportions, and functionality (to an extent) of a 9-month old infant. When its construction and coding are completed, it will function as a “test-bed for theories of social learning (11).” Currently, Nico is far from complete. He lacks a face and therefore expressions; he has limited mobility and has been built and trained to respond to animacy and prosody, but these along with many other features necessary for human detection and interaction have not been fused (11).

The core of this project aims to get Nico to manipulate a naïve adult, through a small set of self-created gestures inspired by Baby Sign, to complete a complex, arbitrary task which requires the completion of multiple steps (actions) or stages. As the experiment progresses, Nico will execute certain motions in response to his visual perception of the subject’s progress in terms of completing the task. The goal is to explore robot-human communication in hopes of getting a better understanding of what kinds of gestures and feedback humans respond to. Furthermore, the purpose is to gage if Nico is successful at manipulating a person at all, or if a baby/robot needs a more comprehensive body or gesture set in order to successfully manipulate an adult’s activities. The study presented in The Art of Manipulation therefore aims to help us learn to further classify a social from a non-social object with the long-term goal of incorporating such knowledge into Nico.

IV. Procedure

IV.A Designing the Project

IV.A.1 - Picking of Motions

The first task I performed was compiling the small set of motions for Nico to execute instruction and feedback. I began by gathering a general set of Baby Signs that were consistently found in Baby Sign dictionaries. Since Nico only had one arm, no face or facial expression, and no ability to consume food or beverages, I then eliminated Baby Signs due to their complexity (for example, signs involving two hands). The final set of
motions was comprised of slightly altered or recreated Baby Signs meant to indicate “yes,” “no,” “again,” “success,” and “pointing (to communicate desire or instruction) (5).”

Yes – Nico’s head starts from center and “nods” head up, then head down. Nico’s head always returns to center.

No - Nico’s head starts from center and “shakes” head left, and then head right. Nico’s head always returns to center.

Again – Combination of 3 “No’s”, pointing at person, pointing under table, and doing the “lower” arm movement (see Appendix B)

Success – Combination of pointing arm straight up, and simultaneously bending elbow and lowering shoulder.²

Pointing – Any straight armed motion that only involves Nico’s shoulder motors to move and is used to indicate a person or a place.

IV.A.2 - Designing the Task

In order to establish that Nico and the test subject were actively engaged in a social interaction, the task had to be simple enough for Nico and QNX, the operating system it runs on, but complicated enough that the initial state and goal state for the task were substantially different so the data was not a representation of a person completing a task because of an overtly obvious goal state. To further prevent this hazard, I decided the task would have to require multiple stages or states that a person would have to pass through to succeed. This would also allow Nico more opportunity to give feedback. The previous constraints allowed me to settle on the task of Nico instructing and guiding someone placed in front of him to get under a table that was within Nico’s range of sight but far enough away that it blended in with the rest of the lab paraphernalia.

In order to accomplish this, the task was divided into 4 stages: the initial stage, the “being-at-table” stage, the “crouched” stage and the goal stage. The only constraint on the order of progression through the stages was to start at the initial stage and end at the goal stage.

Initial Stage – Nico’s head is centered and up, person is standing in front of and about 50cm back from Nico’s table. The subject is wearing a red/orange wrap on their head.

“Being-at-Table” Stage – The subject is within the correct distance of the table to be considered at or close enough to it.

“Crouched” Stage – The subject has lowered their body and their head enough to be considered low enough to be (or eventually be) under the table.

Goal Stage – The subject is in stages “Being-at-Table” and “Crouched” simultaneously, indicating they are (or close enough) under the table.

² This was originally going to be “clapping,” mounting a cardboard arm on Nico and clapping the actual arm into it, but another arm was added.
IV.B - Coding and Implementation

IV.B.1 – Overview

IV.B.1.A – SOFTWARE

Nico is run through the operating system QNX, which can be logged into remotely through ssh kronos.cs.yale.edu. Coding through kronos or on the machines themselves can only be done in the text editor vi. Two computers are needed to compile and run Nico; one is responsible for coding and executing, the other for making files. I also used LINUX machines in the zoo to code in Emacs and bidirectionally transfer my code when necessary.

IV.B.1.B – HARDWARE

Nico consists of a head, a torso, a working arm and a hanging arm. The head has six motor nodes for pitching the head, rolling the head, and shaking the head side to side. Two nodes control the eye camera motor movements, and the shoulder, and elbow each have a node for rolling and a node for pitching. Combined, these motor nodes can create a wide range of movement. Nico also has two non-motor nodes, one for each eye camera (8).

Two computer nodes are required to run any script on Nico. Within each computer are a set of nodes as well and they determine what can and cannot be run, executed or compiled. In order to move data from various nodes, a porter system and a newer, faster, TCP/IP system can be used (11).

IV.B.1.C – HARDWARE

Coding was done on a variety of different nodes, text editors, and operating systems. The original headinit.cc was coded on the “old” nodes which were replaced by “new” nodes with ways of connecting paths, different folders, and different execution of code when compared to execution on the old nodes. On the new nodes, headinit.cc no longer compiled so a lot of code was recreated.

Coding involving the evaluation of perceptual input was originally done through Emacs on the Zoo Linux machines to minimize the time I spent with Nico because of the heavy traffic of other undergraduates and graduate students using the machine. I created two programs, Tasks.cc and Depth.cc. The former simulated the 2-Dimentional inputs my master program, an adaptation of headarm.cpp on the Kronos machine, would receive through TCP/IP by randomly generating x and y values for simulated-input and doing a variety of calculations based on the person’s previous coordinates that would generate numbers corresponding to the type of feedback those values would generate. Those numbers were to then be linked up to the motors in headarm.cpp.

Depth.cc is a program that, after manually calculating the x and y coordinates of two blocks bounding the edge of a table (and doing it for both the image from the right eye and the image from the left eye), calculates the depth of the blocks based on binocular disparity. This was coded out of nervousness that small movements of the table I set up would alter its position in Nico’s frame and would take a very long time to recalculate. It also allowed me to establish a depth for objects whose distance from Nico was great enough that depth could not be calculated or was done incorrectly by the code on Kronos.

Headarm.cpp served as the chief executive program that all data was funneled through. It served as the last end of a series of pipelines; the first, newgrabber.cc got in
visual input from each eye and sent the images to color.h and color.cc which took the images and made boxes bounding a previously hard-coded color. These images were then filtered through newdepth.cc which determined the binocular disparity and resulting coordinates of the boxes in 3-Dimensions; those coordinates were then sent to headarm.cpp for processing. Headarm.cc functioned as a feedback loop; it both used the image data to control motors and decided when to get new image data.

**IV.B.2 – Motors**

**IV.B.2.A – HEAD MOTORS**

All motor movements were hard-coded before integrating perception. The initial “No” and “Yes” were coded on kronos in /home/ebb23/workspace/headinit/headinit.cc. They each did the complete motion 3 times. “No” and “Yes” were updated in headarm.cc in the function HeadReinforcements which combined the previously separated functions for “Yes” and “No” and instead accepted two parameters, one determining whether to shake or nod, and the other determining how many iterations of each motion to execute.

**IV.B.2.B – ARM MOTORS**

I attempted to derive trajectory paths from running previously defined command-line arguments for arm-motor motion, but the nodes crashed often and the command-line call rarely responded in any consistent fashion. The same calls generated different results or different calls would generate the same results. Since there was no consistency, arm motion could not be hard-coded.

**IV.B.3 – Perception**

**IV.B.3.A – TRACKING**

In order to gain any information about the location of the subject relative to the desired goal location of the subject, I decided that I would wrap a bright red fleece around the subject’s head and use a crude tracking system of the red fleece color moving by specializing in color.h the specific combinations of red, green and blue that would allow to indicate the red range it was to look for and positively identify and box (Tasks.cc). Before tracking began, the location of the table was calculated. To locate the table, two red blocks were placed on the opposing edges of the table, and the image from camR and camL was saved. These pictures were then opened in Paint and the location, in pixels, of the bounding boxes in each image was manually recorded. These were then fed to Depth.cc which generated the slope of the table edge and its depth, which could now serve as the frame of reference for tracking and goal-directing. The blocks were then removed.

Tracking involved keeping the old coordinates of the bounding box for the red fleece, its current coordinates, and calculating through the distance formula if a person moved closer to or farther from the goal. Feedback was generated based on the amount of improvement made at each calculation. If completion is detected, Nico was to indicate success and stop processing the task tracking. Tasks.cc also puts a limit to how many iterations the system can run before it declares a task-completion failure.
IV.B.3.B – Path

In order to run headarm.cpp, a huge pipeline was generated to convert the raw images picked up by the cameras in Nico’s eyes to an array of structures containing coordinates for each bounding-box of the specific, pre-specified color of interest. Newgrabber.cc took raw images from each eye camera separately at ~9.8 frames/s. This information was portered (TCP/IP on new nodes) to cam.cc, which generated image boxes on the screen, one for each eye. These images were then sent to color.cc and color.h where every blob of color within the pre-specified range was tagged with a box around it, a center point and the structure was added to an array. Color.cc then piped this array to newdepth.cc which resolved the binocular disparity between the right camera and the left camera, and gave each box an x, y, and z coordinate. At most, the two largest x, y, z structures were added to a new array which was sent to headarm.cc, prepared for 3-D processing.

V. Results/Discussion

In the process of implementing the explicit design structure for this project, I ran into numerous unexpected difficulties. The design of the project had been formed on a variety of assumptions about Nico: his functional capabilities, his perceptual capabilities, and his consistency in implementing coded functional and perceptual tasks. Such assumptions were necessary in order for Nico to successfully interpret and direct behavior to completion (or a logical progression of movement) in the “Get-Under-Table” task. However, realization that certain assumptions about Nico were incorrect made the task improbable if not impossible to complete.

The first problem emerged from the task design itself. In order for any feedback to be registered as influential to the course of the project, the task had to be arbitrarily difficult; if the goal stage were too obvious, the robot feedback would not generate communication. To accomplish this, the table had to be placed a significant distance away from Nico (~2 meters) so the initial thoughts of a naïve subject would not be automatically biased. Newdepth.cc porters x, y and z coordinates of blobs of color into headarm.cc after making depth calculations from 2-D images. However, Nico’s eye cameras are designed for focusing on much shorter distances. In calculating binocular disparity and depth, if an object is too far away, its depth becomes nearly impossible to calculate. In order to determine if a person completed the task, its x, y and z coordinates as well as those of the table had to be of logical, usable value. They were not. As the person approached the table, the binocular disparity between the two eyes was so small that all three coordinates could not accurately be calculated; thus the person could not be accurately directed nor could Nico determine if the person were in their goal state.

In an attempt to solve this problem, I altered the position of the table and brought it closer to Nico. The distance of the table from Nico that worked the most efficiently placed it far too close to the actual starting position of the person and introducing bias. The second option was to move the table more peripheral to Nico’s eyes but keep its distance within a short range. Nico only has a limited window and the peripheral distance one could move the table was limited by that. The last table trial was to put it outside of the frame of vision and calculate a pixel-to-actual distance conversion for mapping location. This introduced two large projects, head tracking (or fleece tracking) by Nico and searching for the table position at the beginning of each run. It also required
all relative distances to be converted to actual distances and to track overall as well as relative movement with respect to transitions between each frame. These projects alone were enormous. I also attempted to manually calculate depth and distance with Depth.cc, but did not get any significant results. Therefore the first major difficulty stemmed from Nico’s visual limitations.

The next assumption I made was of the consistency of the motors. For example, given a trajectory to a point as an argument and then given the reverse trajectory, I expected that Nico would return to his original position before the trajectory-moves were called. Instead, the arm and the head motors demonstrated incredible inconsistency in their functionality. The biggest problem involved Nico’s NECK_YAW head motor. Giving Nico equal but opposite commands in order to generate a “head shake no” motion did not return Nico’s head to its previous destination. The degree by which Nico was off from center was also not consistent, so I could not compensate by adding an extra movement command.

Since Nico has no sense of his own inner state, it is impossible to gauge where his cameras were picking up information from. The position of the table could not be guaranteed to remain constant, making it impossible for correct goal direction. Sometimes the movements of Nico’s head were so drastic that the blob and the table were absent all together, giving Nico no frame of reference to return to. Head tracking could find the person’s head, but there was no way to account for the possible disparity in head motion. Shaking was the central negative feedback command and without it, giving negative feedback was impossible or would involve implementing a different form of negative feedback which would arguably be more complex than “nodding,” which would skew the data and contradict the purpose of analyzing communication through simple motion.

In order to give the initial command and to signify completion, arm specified commands were to be used. The arm commands were controlled by various motors that, similar to the NECK_YAW motor, were completely inconsistent in their response to identical commands. In order to test the arm movements, I ran the functional headarm.cc out of Marek’s directory to ensure that my code had not affected the function of my robot arm. Both headarm.cc files in separate directories produced the same types of errors. The arm would switch between not responding to motion calls, giving the same response for different calls, or give different results for identical calls. Since the program “slider” was not transferred in a functional state to the new nodes, there was no way to map a trajectory and read it from a file, which seemed to generate more consistent behavior. In conclusion, the arm could not be implemented to perform basic movements.

Between variable generation of video footage, depth footage, and node function, Nico demonstrated that the system on which he runs and the type of motor and perceptual variability it that system introduces, my chosen task was out of the scope of Nico’s capabilities. A different task choice would possibly have yielded different results, but the types of perceptual analysis and the required motor consistency led to the errors in the completion of the task-feedback system.
VI. Conclusions

The initial goal of this project was to evaluate human communication so as to better clarify what qualities are the most valuable and important in non-verbal communication. The desire was to evaluate non-verbal communication through the success or failure of the impact of Nico’s simple actions and to attempt to tease out a correlation between the two. However, the underlying expectation I had was that Nico’s behavior would be based on a logically defined set of code for both control and interpreting perceptual feedback into logical motor feedback. Instead, Nico demonstrated that before considering motions or behaviors, the logical consistency of a system must be established; otherwise there is no possibility of making a social interaction. In a sense, I became the test subject whose goal, to code Nico, was shaped and defined by Nico’s lack of consistency.

Social behavior is a complex combination of many factors; expectation, logic and consistency are of central importance. In my experiment, the expectations were that the computer would function in a binary, logical fashion, that Nico would respond to this in a logical and consistent fashion and that the human would also elicit logical responses to Nico’s behavior. Yet when the consistency in Nico breaks down, the behaviors exerted by him no longer represent logic but randomness. Thus, the human would not be able to interact with and establish the social connection. Behavioral influence of motions can only be analyzed if these expectations are predominantly adhered to. A breakdown in such a connection prevents this type of experiment from generating useful results.

With computers and humans as well, a person builds their trust and relationship based on the perceived feedback of a system to a given situation. What this experiment demonstrates is that when the system generates feedback that cannot be logically understood, a relationship cannot form. Though human behavior is comprised of too many factors to justify that it always responds in a logical fashion, consistency in behavior helps remove “noise” and allows for understanding, one of the aspects of a social interaction (10). In conclusion, though I expected to analyze communication through the analysis of actions, I instead learned that on a more basic and important level, communication involves building a relationship largely defined by logic and consistency. In order to make a truly social robot, it must be logical (to a good extent) but most importantly consistent. Otherwise, relating socially to a semi-amorphous combination of metal and nodes will be impossible.

VII. Future Implications

The future implications of this project are numerous. It introduces another way in which human behavior can be defined, which presents a new series of questions. In order to fully analyze a robot in the context the original experiment intended, one must first pick a philosophical viewpoint on how behavior gets defined, how much is rooted in logic, how much is rooted in emotion, how much is rooted in expectation and how they are intertwined. Computers provide an example of a system that is supposed to run completely logically; decisions are 1 or 0, and there is no in between. Yet when a system like that fails, our behaviors are affected; our emotions change and our expectations are challenged. It forces one to rectify to situations that cannot possibly coexist. One theory that psychology and cognitive psychology struggle with is how one defines the mind; is the mind a sum of its functional parts or is it a state that cannot be directly linked to
circuitry? This project presents a challenge to the possibility of creating a social robot, and if it is, how one defines and justifies that a robot is truly social. Perhaps it is a 2006 version of the Turing test.

However, assuming that making a social robot is possible, this experiment, when successfully completed and expanded, can help define what components are necessary and most effective for communication. It will further Nico's movement toward being a completely social robot. Possible studies with autistic patients can be done to learn what they respond to and if there is a discrepancy between the autistic and our experimental population. Non-verbal communication is incredibly powerful and a constant part of life. Truly understanding the underpinnings can help us as humans better understand ourselves and each other as well.

VIII. Acknowledgements
Thank you to the following people for your guidance, time and patience:

Kevin Gold
Marek Doniec
Ganghua Sun
IX. References


APPENDIX A
Examples of Baby Sign Language

*This Appendix contains names of Baby Signs and occasional picture representations of the actual sign being done. These signs are drawn from two libraries.

Dictionary from “Signing with Your Baby” with selected examples (2)

Again

Airplane
All Gone
All Done
Alligator
Ape
Apple
Baby
Ball
Bath
Banana
Bird
Biscuit
Book
Bread
Butterfly
Candy
Car
Cat
Change

Chase
Cheese
Clown
Cold
Cow
Cracker
Cry
Daddy

Dirty
Dog
Down
Duck
Eat

Elephant
Elk
Fish
Frog
Hat
Helicopter
Help
Horse
Hug

**Hurt**

Medicine
Milk
Mommy
Monkey
More
More and Eat
Music
OK
Play
Please

Shoes
Sleep
Smile
Table
Toothbrush
Thank You
Dictionary Examples from “Baby Signs and Baby Signing the Joseph Garcia Way (3)”
Sign for “please”
APPENDIX B
Working Dictionary of Movements for Nico

*This is a dictionary describing simple arm and head movements and complex movements derived from combining more simple movements.

**Arm down** – all nodes set to put arm at Nico’s side
**Arm wide** – arm straight, coming out sideways, uses only shoulder nodes…can give it an angle.
**Arm forward** - arm straight, coming out forwards, uses only shoulder nodes…can give it an angle.
**Elbow r_in** – Rotate elbow node so lower arm goes clockwise
**Elbow bend in** – Use non-rotating elbow node, negative trajectory (brings hand to upper arm)
**Elbow bend out** – Use non-rotating elbow node, pos trajectory until arm straight (brings arm to straight from bent.
**Arm bend up**– straight arm down, only 2 elbow nodes move hand to shoulder
**Arm bend down**– straight arm down, only 2 elbow nodes move hand away shoulder
**Arm bend twist in** – arm bend up+ use (elbow) shoulder nodes to twist lower arm in towards body
**Arm bend twist out** – arm bend up + use (elbow) shoulder nodes to twist lower arm away from body
**Wrist Flex** –wrist nodes isolated, bend hand toward arm
**Wrist left** – wrist arch + leaning left
**Wrist right** – wrist arch + leaning right
**Wrist Arch** – same as beckon, wrist nodes move hand away from arm
**Wrist Beckon** – wrist flex + wrist arch, (x3)
**Wrist Swoop** – wrist normal, then wrist left with up trajectory, then wrist right with down-up trajectory
**Arm point** – arm wide + arm forward (deg?)
**Excited arm point** – arm point + shoulder nodes making miniscule up and down motions (2degrees?)
**Alternate excited arm point**- arm point + arm bend (maybe twist in) back and forth, fast (x3?)
**Wave movement** – arm twist out + arm twist in (x3?)
**Touch mouth** – arm forward + arm bend twist in (so hand touches mouth-stickers)
**Cover Eyes** – Arm sideways + arm bend twist in (so hand covers eyes-stickers)
**Make noise** – Touch mouth + elbow and maybe hand straighten in slightly down-up trajectory
**Lower** – Arm starts at side, elbow bends toward upper arm, both shoulder rolls up and forward as elbow straightens. Wrist Arch. Shoulder drops sideways up to down, elbow bends to arm until shoulder rotation places elbow below shoulder and elbow then straightens, move wrist from arch to flex at same rate as shoulder dropping.
**Success** – Arm straight up. Elbow r_in. Simultaneous: Arm forward (downward from sky to floor) + elbow bend in. Then Simultaneous: Arm forward (up from floor to sky) +
elbow bend out. Arm Down (all nodes return arm to side). Head Down (-45, it should be center at 0)

Maestro - arm forward, elbow bent up, wrist swoop
Super Maestro – Maestro + arm left (if wrist left) arm right (if wrist right), little trajectory w/wrist

Move object – arm point at object + arm point at new location
Sing – Touch mouth + Make Noise + Maestro (prosody detection)

Turn off lights – Point to lights + Cover eyes + excited arm point to light

Bounce a ball – point to a ball, arm forward, arm bend up + arm bend down (x3?)

animacy

Hide under a desk- Point to person, point to desk, arm bend up + arm bend down (x3?) gradually getting lower with hand

Clap – twist arm in against prosthetic, make clapping noises

Sit- point to chair, arm forward all the way up, wrist, then all the way down +wrist arch

Stand – (have to be sitting) flip hand, wrist flex, elbow bend up, wrist progressively more arched till arm straight up, elbow straight, wrist arched

Put on a hat – point to object, point to part of person’s body

Dance – music + Nico’s arm going up and down sideways (or front), music stops, point at person, start music. (animacy detection)
APPENDIX C
Key Code Files

*This Appendix contains three important code files used to make, run and test data for my project. In includes:

1) headarm.cpp – Responsible for receiving, integrating and transforming perception data into motor commands.
2) Tasks.cc – Simulator for 2-D perceptual system
3) Depth.cc – Functionally calculates depth from hard coded binocular coordinate sets

Headarm.cpp
/*
headarm.cpp in ebb23@kronos/workspace/headarm/headarm.cpp
Head-arm coordination
m_set_group_addr(motor_no, group_addr, is_leader) !!!!
**************Erica’s Comments***************
This served as the central executive program that manipulated motor data, interpreted perceptual data and integrated the two processes. It is the end of a series of pipelines, converting raw 2-D video footage to 3-D image representation by portering data into various programs and portering out result computations through TCP/IP.
*/
#include <time.h>
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
#include <math.h>
#include <vector>
#include <time.h>
using namespace std;
#include "libmotor.h"
#include "Timestamp.h"
#include "newdepth.h"
#include "newgrabber.h"
#include "cam_data1.h"
#include "cam_data2.h"

#include "TcpIpImage.h"

//Process name
char *my_name = "/headarm";
TcpIpReceiver<Coord3Ds> receiver_Depth_Info;
TcpIpSender<ClipInfo> sender_ClipInfoR;
TcpIpSender<ClipInfo> sender_ClipInfoL;
TcpIpSender<RecCommand> sender_RecCommandR;
TcpIpSender<RecCommand> sender_RecCommandL;

//InputPortOf<Coord3Ds> InDepth_Info;
//OutputPortOf<ClipInfo> OutPortClipInfoR;
//OutputPortOf<ClipInfo> OutPortClipInfoL;
// OutputPortOf<RecCommand> OutPortRecCommandR;
// OutputPortOf<RecCommand> OutPortRecCommandL;

// The ratio of motor servo counts to seconds, # of encoder ticks per revolution
#define SPEEDRATIO 1953
#define ENCODERRATIO 2048
#define HEAD_BOARD 0
#define ARM_BOARD 1
#define BOTH_BOARDS 2
#define NUM_MOTORS 15
#define HEAD_MOTORS 6
#define HEAD_MOTOR_BASE 0
#define ARM_MOTORS 6
#define ARM_MOTOR_BASE 8
#define HAM_CODE 4
#define NECK_ROLL 1
#define NECK_YAW 2
#define UPPERNECK_PITCH 3
#define EYE_PITCH 4
#define RIGHT_PAN 5
#define LEFT_PAN 6
#define SHOULDER_PITCH 9
#define SHOULDER_ROLL 10
#define UPPERARM_ROLL 11
#define UPPERARM_PITCH 12
#define LOWERARM_ROLL 13
#define LOWERARM_PITCH 14
#define LOWERNECK_PITCH 15
#define TRAJ_INI_DELAY 500
#define TIMEOUT 10000
#define GROUP_END -1

#define RANDOM_POSE_MAX 500
int Num_Random_Pose=500;
#define VIDEO_PATH "/tmp/video/
#define PATHS_PATH "/home/marek/paths/

// Calculated from the xCoor, left to right, and the z_coors, toward and away from nico
#define Table_Slope 0.19565
#define Table_b 138.44613
#define VIDEO_FRAMES 2100
int gear_ratio[NUM_MOTORS+1]= {1,
    // md 246,246,246,141,1,1,1,1,
    246,246,246,141,1,1,1,1,
    //128,231,86,86*2,246,246,128};
    //Ganghua -- temporarily change ratio[arm_6] ->
    //128,231,86,86*2,246,370*3,128};
    // md 128,231,86,86*2,415,370*3,128};
    128,231,86,86*2,415,134,128};
//long upperlimit[NUM_MOTORS+1]= {0, 30, 30, 30, 15, 0, 0, 0, 0, 60, 20, 15, 90, 60, 60, 0} previous experiment setting 60, 15, 15, 105, 90, 30, 0};

long upperlimit[NUM_MOTORS+1]= {0, 30, 30, 30, 15, 0, 0, 0, 0} previous experiment setting 45, 15, 0, 90, 90, 30, 0};

long lowerlimit[NUM_MOTORS+1]= {0, -30, -30, -45, -15, 0, 0, 0, 0, 0, 90, 90, -30, -45, -15, 0};

long lowerlimit[NUM_MOTORS+1]= {0, -30, -30, -45, -15, 0, 0, 0, 0, 25, -15, -60, 30, 0, -15, 0}; previous experiment setting 15, -30, -60, 00, 0, -15, 0};

int arm_group[ARM_MOTORS+1];
int arm_group_addr=150;
int head_group[HEAD_MOTORS+1];
int head_group_addr=200;

int pose[RANDOM POSE_MAX][ARM COARSE+1];
Coord3D coord[RANDOM POSE_MAX];
float motion_error[RANDOM POSE_MAX][2];

//Ganghua
//LOWERNECK_PITCH, UPPERNECK_PITCH, SHOULDER_ROLL's directions are reverse.

extern int row_size320, col_size320, row_size640, col_size640;

float motor_history[200][ARM_COARSE+1];
int history_counter;

// MAREK: GLOBAL CONFIGURATION VARIABLES

int global_config[100];

#define GCONF_VIDEO 0

void wait_stop_group(int *group)
{
    motor_info minfo;
    int moving=0;
    int index=0;
    int waiting_time=0;
}
delay(TRAJ_INI_DELAY);

while(1)
{
    while(group[index]! = GROUP_END)
    {
        // cout << \"Check motor \" << group[index] << endl;
        m_get_status(group[index], &minfo, 1);
        if(!minfo.move_done)
        {
            // cout << \"Motor \" << group[index] << \" still moving.\" << endl;
            moving = 1;
        }
        index++;
    }
    if(moving && waiting_time<TIMEOUT)
    {
        delay(100);
        waiting_time+=100;
        moving=0;
        index=0;
    }
    else if(waiting_time>=TIMEOUT)
    {
        cout << \"Warning: wait_stop_group time out.\" << endl;
        return;
    }
    else
    {
        // cout << \"wait_stop_group waiting time:\" << waiting_time << endl;
        return;
    }
}

void wait_stop_motor(int addr)
{
    motor_info minfo;
    int waiting_time=0;

delay(TRAJ_INI_DELAY);

while(1)
{
    m_get_status(addr, &minfo, 1);
    if(!minfo.move_done && waiting_time<TIMEOUT)
    {
        delay(100);
        waiting_time+=100;
    }
    else if(waiting_time>=TIMEOUT)
    {
        cout << \"Warning: wait_stop_motor time out.\" << endl;
        return;
    }
    else
    {
        cout << \"wait_stop_motor waiting time:\" << waiting_time << endl;
    }
}
return;
}

void stopAllMotors (int mode)
{
    if(mode==HEAD_BOARD || mode==BOTH_BOARDS)
        for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
            m_stop_smooth(i);
    if(mode==ARM_BOARD || mode==BOTH_BOARDS)
        for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
            m_stop_smooth(i);

    //delay to make sure the motors have actually stopped
    //Ganghua zzzzzzz I am not sure whether 1 second is enough to stop all motors
    delay(1000);
}

void print_motor_pos(int mode)
{
    if(mode==HEAD_BOARD || mode==BOTH_BOARDS)
    {
        cout << "Head motor positions (in encoder unit): ";
        for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
        {
            m_get_pos(i,&pos[i],true);
            cout<<pos[i];
            if(i!=HEAD_MOTORS)
                cout<<",";
            else
                cout<<endl;
        }
        cout << "Head motor positions (in degree): ";
        for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
        {
            m_get_pos(i,&pos[i],true);
            cout<<(float)pos[i]/2048*360/gear_ratio[i];
            if(i!=HEAD_MOTORS)
                cout<<" ",
            else
                cout<<endl;
        }
    }
    if(mode==ARM_BOARD || mode==BOTH_BOARDS)
    {
        cout << "Arm motor positions (in encoder unit): ";
        for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
        {
            m_get_pos(i,&pos[i],true);
            cout<<pos[i];
            if(i!=ARM_MOTOR_BASE+ARM_MOTORS)
                cout<<" ",
            else
                cout<<endl;
        }
        cout << "Arm motor positions (in degree): ";
    }
}
for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
{
    m_get_pos(i,&pos[i],true);
    cout<<((float)pos[i]/2048*360/gear_ratio[i];
    if(i!=ARM_MOTOR_BASE+ARM_MOTORS)
        cout<<","
    else
        cout<<endl;
}

void save_motor_pos(int mode, ofstream & out)
{
    if(mode==HEAD_BOARD || mode==BOTH_BOARDS)
    {
        out << "Head motor positions (in encoder unit): ");
        for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
        {
            m_get_pos(i,&pos[i],true);
            out<<pos[i];
            if(i!=HEAD_MOTORS)
                out<<"",
            else
                out<<endl;
        }
    }
    out << "Head motor positions (in degree): ");
    for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
    {
        m_get_pos(i,&pos[i],true);
        out<<((float)pos[i]/2048*360/gear_ratio[i];
        if(i!=HEAD_MOTORS)
            out<<"",
        else
            out<<endl<<endl;
    }

    if(mode==ARM_BOARD || mode==BOTH_BOARDS)
    {
        out << "Arm motor positions (in encoder unit): ");
        for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
        {
            m_get_pos(i,&pos[i],true);
            out<<pos[i];
            if(i!=ARM_MOTOR_BASE+ARM_MOTORS)
                out<<"",
            else
                out<<endl;
        }
    }
    out << "Arm motor positions (in degree): ");
    for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
    {
        m_get_pos(i,&pos[i],true);
        out<<((float)pos[i]/2048*360/gear_ratio[i];
        if(i!=ARM_MOTOR_BASE+ARM_MOTORS)
            out<<"",
        else
            out<<endl<<endl;
}
void traj_deg(int addr, int degree, bool now) {
    // Ganghua
    if(addr==12)
        m_traj_position(addr, (long)((float)degree/360*2048*gear_ratio[addr]), 40.0, 0.04, now);
    else
        m_traj_position(addr, (long)((float)degree/360*2048*gear_ratio[addr]), 20.0, 0.02, now);
}

void traj_deg(int addr, float degree, bool now) {
    // Ganghua
    if(addr==12)
        m_traj_position(addr, (long)(degree/360*2048*gear_ratio[addr]), 40.0, 0.04, now);
    else
        m_traj_position(addr, (long)(degree/360*2048*gear_ratio[addr]), 20.0, 0.02, now);
}

void return_to_start_pos() {
    for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
        m_traj_position(i, 0, 10.0, 0.02, FALSE);
    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);
}

void return_head_to_start_pos() {
    for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
        m_traj_position(i, 0, 10.0, 0.02, FALSE);
    m_start_move(head_group_addr, FALSE);
    wait_stop_group(head_group);
}

int get_kine_mapping() {
    ifstream motordata("motordata.dat");
    ifstream objectdata("objectdata.dat");

    int i;
    for(i=0; i<Num_Random_Pose; i++) {
        for(int j=1; j<=ARM_COARSE; j++)
            motordata >> pose[i][j];
        objectdata >> coord[i].x;
        objectdata >> coord[i].y;
        objectdata >> coord[i].z;
        objectdata >> coord[i].xR;
        objectdata >> coord[i].yR;
        objectdata >> coord[i].xL;
        objectdata >> coord[i].yL;

        if(objectdata.eof())
            break;
    }
}
motordata.close();
objectdata.close();

cout << "Kinematics mapping learnt." << endl;
cout << i << " mappings have been retrieved." << endl;
return i;
*/
int read_depth(Coord3D* coord, int turn_waist)
{
    traj_deg(14,0,true);
    wait_stop_motor(14);
    int angle=0;

    while(abs(angle)<=20)
    {
        if(InDepth_Info.Read(0) && !delay(500) && InDepth_Info.Read(0))
        {
            *coord=InDepth_Info.Content();
            if(coord->z>100)
                return 0;
            else if(turn_waist)
            {
                if(coord->z==1)
                {
                    angle+=10;
                    traj_deg(14,angle,true);
                    wait_stop_motor(14);
                }
                else if(coord->z==2)
                {
                    angle=10;
                    traj_deg(14,angle,true);
                    wait_stop_motor(14);
                }
                else
                    break;
            }
            else
                break;
        }
        else
            break;
    }
    traj_deg(14,0,true);
    wait_stop_motor(14);
    return 1;
} /*
int read_depth(int mode, Coord3D* coord, Coord3D* target)
/*
mode = 0 fill coord with the only object information from InDepth_Info
mode = 1 fill target with the only object information from InDepth_Info
mode = 2 fill coord with the object information different from target
*/
{
int ret_value=0;
float target_tolerance=50.0;
int to_fill=-1;

Coord3Ds depth_info;

if(!(receiver_Depth_Info.Read(&depth_info)) && !delay(500) && !(receiver_Depth_Info.Read(&depth_info)))
{
    //Coord3Ds & depth_info = InDepth_Info.Content();
    //
    // float z1 = (depth_info.coords[0].z > depth_info.coords[1].z ? depth_info.coords[0].z :
    // depth_info.coords[1].z);

    switch(mode)
    {
        case 0: if(depth_info.num==1)
        {
            coord->Copy(depth_info.coords[0]);
            else
            ret_value=1;
            break;
        case 1: if(depth_info.num==1)
        {
            target->Copy(depth_info.coords[0]);
            else
            ret_value=1;
            break;
        case 2: // cout << "mode 2" « endl;
        if(depth_info.num==2)
        {
            // cout << "num=2" « endl;
            float dist0=target->Distance3D(depth_info.coords[0]);
            float dist1=target->Distance3D(depth_info.coords[1]);
            // cout << dist0 « endl;
            // cout << dist1 « endl;
            if(dist0 < dist1 && dist0 < (target_tolerance * (depth_info.coords[0].z / 300)) )
            to_fill=1;
            else if(dist1 < dist0 && dist1 < (target_tolerance * (depth_info.coords[1].z / 300)) )
            to_fill=0;
            else
            {
                ret_value=1;
                break;
            }
            coord->Copy(depth_info.coords[to_fill]);
            target->Copy(depth_info.coords[1-to_fill]);
        }
        else if (depth_info.num == 1)
        {
            // cout << "num=1" « endl;
            coord->Copy(depth_info.coords[0]);
        }
        else
        {
            // cout << "num=??" « endl;
            ret_value=1;
        }
        break;
    default: ret_value=1;
    break;
    }
}
```c++
else
    ret_value=1;

return ret_value;

void spline(int npoints, int dim, int makepoints, long** src, long** dest) {
    cout << "SPLINE Interpolation called. Processing " << npoints << " points in " << dim << " dimensions. " << endl;
    cout << "Resulting array will contain " << makepoints << " points." << endl;
    float B[npoints];

    for (int x = 0; x < makepoints; x++) {
        float t = ((float)x / (float)(makepoints - 1));
        // cout << "t:" << t << " \n";
        for (int i = 1; i < npoints; i++)
            B[i] = 0.0f;
        B[0] = 1.0f;

        for (int j = 0; j < npoints - 1; j++)
            for (int i = j; i >= 0; i--)
                B[i + 1] += t * B[i];
        B[i] = (1 - t) * B[i];
    }

    for (int k = 0; k < dim; k++)
        dest[k][x] = 0;

    for (int i = 0; i < npoints; i++)
    {
        testi += B[i];
        cout.precision(3);
        cout << "[" << i << "]=" << B[i] << " \n";
    }
    cout << "sum=" << testi << endl;
}

void bspline(int npoints2, int dim, int makepoints, int degree, long** src, long** dest) {
    int npoints = npoints2 - 2;

    cout << "BSPLINE Interpolation called. Processing " << npoints2 << " points in " << dim << " dimensions using degree " << degree << " polynomials." << endl;
    cout << "Resulting array will contain " << makepoints << " points." << endl;

    float NA, NB;
    float N[npoints + degree + 1];
    float T[npoints + degree + 1];
```
float dist[npoints];
float totaldist = 0.0f;

for (int i = 0; i < npoints2 - 1; i++) {
    float tmp = 0.0f;
    for (int k = 0; k < dim; k++) {
        tmp += (((float)(src[k][i + 1] - src[k][i])) * ((float)(src[k][i + 1] - src[k][i])));
    }
    dist[i] = sqrt(tmp);
    totaldist += dist[i];
    cout << "dist[" << i << "]=" << dist[i] << "  totaldist=" << totaldist << endl;
}

// initialize timeslices

if (degree % 2) {
    for (int i = 0; i < npoints + degree + 1; i++)
        T[i] = ((float)i / (float)(npoints + degree - 1));
} else {
    float tsum = 0.0f;
    for (int i = 0; i < (degree / 2); i++) {
        T[i] = tsum;
        tsum += (2 * dist[0] / (totaldist * (float)degree));
    }
    for (int i = 0; i < npoints; i++)
        T[degree / 2 + i] = tsum;
        tsum += (dist[degree / 2 + i] / totaldist);
    for (int i = 0; i < (degree / 2); i++)
        T[i + npoints + degree / 2] = tsum;
        tsum += (2 * dist[npoints - 2] / (totaldist * (float)degree));
    T[npoints + degree] = 1.0f;
}

for (int i = 0; i < npoints + degree + 1; i++)
    T[i] = ((float)i / (float)(npoints + degree - 1));

cout.precision(3);
for (int i = 0; i < npoints + degree + 1; i++)
    cout << i << "= " << T[i] << ",";

for (int x = 0; x < makepoints; x++) {
    float t = ((float)x / (float)(makepoints - 1));
    cout << "t: " << t << ",";
    for (int i = 0; i < npoints + degree + 1; i++)
        if ((T[i] <= t) && (t < T[i + 1]))
            N[i] = 1.0f;
        else {
            N[i] = 0.0f;
        }
    // for (int i = npoints; i < npoints + degree + 1; i++)
    //    N[i] = 0.0f;
for (int k = 1; k < degree; k++) {
    for (int i = 0; i < (npoints + degree + 1 - k); i++) {
        $N[i] = N[i] * (t - T[i]) / (T[i + k] - T[i])$
        $+ N[i + 1] * (T[i + k + 1] - t) / (T[i + k + 1] - T[i + 1]);$
    }
}

float sum = 0.0f;
for (int i = 0; i < npoints; i++) sum += $N[i]$;
if (t < T[2]) {
    NA = 1.0f - sum;
} else {
    NA = 0.0f;
}
if (t >= T[npoints]) {
    NB = 1.0f - sum;
} else {
    NB = 0.0f;
}

/*
   cout.precision(3);
   cout << "NA=" << NA << "  
for (int i = 0; i < npoints; i++) {
    cout << "[" << i << "]=" << N[i] << "  
}
   cout << "NB=" << NB << "  
   cout << "sum=" << sum << endl;
*/
for (int k = 0; k < dim; k++) {
    dest[k][x] = 0;
    dest[k][x] += (long)(NA * (float)src[k][0]);
    dest[k][x] += (long)(NB * (float)src[k][npoints2 - 1]);
    for (int i = 0; i < npoints; i++)
        dest[k][x] += (long)(N[i] * (float)src[k][i + 1]);
}
}

void move_on_path(int path_points, long **path, int dim, int *group, int group_addr) {
    int path_counter = 0;
    int done = 0;
    int add_n = 0;
    byte npoints;
    cout << "Sending " << path_points << " point-path to motorboards." << endl;
    while ((path_counter < path_points) && (path_counter < 90)) {
        add_n = path_points - path_counter;
        if (add_n > 7) add_n = 7;
        for (int j = 0; j < dim; j++) {
            m_add_path_points(group[j], add_n, (path[j]+path_counter));
            delay(5);
        }
        path_counter += add_n;
    }
    cout << "Send " << path_points << " path points before start. Starting path now!" << endl;
m_start_path(group_addr, group[0]);

while (!done) {
    m_get_npoints(group[0], &npoints, true);
    cout << "npoints = " << (int)npoints << " path_counter = " << path_counter << endl;
    if (npoints == 0) {
        cout << "Buffer underrun in motorboard! Aborting..." << endl;
        done = 0;
    }
    if (npoints < 60) {
        while ((path_counter < path_points) && (npoints < 90)) {
            add_n = path_points - path_counter;
            if (add_n > 7) add_n = 7;
            for (int j = 0; j < dim; j++) {
                m_add_path_points(group[j], add_n, (path[j] + path_counter));
                delay(5);
            }
            path_counter += add_n;
            npoints += add_n;
        }
        if (path_counter == path_points) done = 1;
    } else {
        delay(250);
    }
}

cout << "Path sent, wait to stop: [";

while (npoints > 0) {
    m_get_npoints(group[0], &npoints, true);
    delay(80);
    cout << "*
    cout.flush();
}

cout << "] - Path done!" << endl;

long deg_to_pos(float degree, int addr) {
    return (long)((float)degree / 360.0f * 2048 * gear_ratio[addr]);
}

float pos_to_deg(long pos, int addr) {
    return (float)((float)pos*360.0f / (2048.0f * (float)gear_ratio[addr]));
}

long **new_path_array(int size, int dim) {
    long **path = new long*[dim];
    for (int i = 0; i < dim; i++)
        path[i] = new long[size];

    for (int i = 0; i < size; i++)
        for (int j = 0; j < dim; j++)
path[j][i] = 0;

return path;

}

void delete_path_array(long **path, int dim) {
    for (int i = 0; i < dim; i++)
        delete path[i];
    delete path;
}

long **load_path_from_file(char *filename, int & size, int & dim) {
    size = 0;
    dim = 0;

    ifstream pathfile(filename);
    pathfile >> dim;
    pathfile >> size;

    long **path = new_path_array(size, dim);

    for (int i = 0; i < size; i++) {
        for(int j = 0; j < dim; j++) {
            pathfile >> path[j][i];
            // cout << path[j][i] << ", ";
        }
        // cout << endl;
    }
    pathfile.close();

    return path;
}

long **load_deg_path_from_file(char *filename, int & size, int & dim, int *group) {
    size = 0;
    dim = 0;

    ifstream pathfile(filename);
    pathfile >> dim;
    pathfile >> size;

    long **path = new_path_array(size, dim);

    float tmp;

    for (int i = 0; i < size; i++) {
        for(int j = 0; j < dim; j++) {
            pathfile >> tmp;
            path[j][i] = deg_to_pos(tmp, group[j]);
            // cout << tmp << ", ";
        }
        // cout << endl;
    }
    // cout << endl;
}
pathfile.close();

return path;
}

void save_path_to_file(char *filename, long **path, int size, int dim) {

    ofstream pathfile(filename);
    pathfile << dim << " ";
    pathfile << size << endl;
    for (int i = 0; i < size; i++) {
        for(int j = 0; j < dim; j++)
            pathfile << path[j][i] << " ";
        pathfile << endl;
    }
    pathfile.close();
}

void load_object_pos(char *filename, Coord3D & goal) {

    ifstream oposfile(filename);
    oposfile >> goal.x;
    oposfile >> goal.y;
    oposfile >> goal.z;
    oposfile.close();
}

void save_object_pos(char *filename, Coord3D & goal) {

    ofstream oposfile(filename);
    oposfile << goal.x << " " << goal.y << " " << goal.z << endl;
    oposfile.close();
}

long time_msec() {
    struct timespec tnow;
    clock_gettime(CLOCK_REALTIME, &tnow);
    return (tnow.tv_sec * 1000L) + (tnow.tv_nsec / 1000000L);
}

int construct_new_mapping_old() {
    int pose_index=0;
    int num_trials=0;
    float random_number;
    //float dist;
    srand((unsigned int)(Timestamp::NowAsSeconds ())); 

    ofstream motordata("motordata.dat");
    ofstream objectdata("objectdata.dat");
    ofstream posdata("posdata.dat");

    //Ganghua
    //Range of motion in degree.
    long motion_range[NUM_MOTORS+1];
for(int i=1; i<=NUM_MOTORS; i++)
    motion_range[i]=upperlimit[i]-lowerlimit[i];

int randomize_this = 0;

while(pose_index<Num_Random_Pose)
{
    if (randomize_this) {
        randomize_this = 0;

        for(int i=1; i<=ARM_COARSE; i++)
        {
            random_number=(float)rand();

            /*
             * if(pose_index==0)
             * {
             * */
            pose[pose_index][i]=(int)(random_number/RAND_MAX
                           *(upperlimit[i+ARM_MOTOR_BASE]-lowerlimit[i+ARM_MOTOR_BASE])
                           +lowerlimit[i+ARM_MOTOR_BASE]);

             /*
             * if(i==3)
             *      pose[pose_index][i]=-45;
             * else if(i==4)
             *      pose[pose_index][i]=80;
             * else if(i==1)
             *      pose[pose_index][i]=30;
             * }
             * else
             * {
             *      pose[pose_index][i]=pose[0][i];
             * }
             */

        }
    }

    for(int i=1; i<=ARM_COARSE; i++)
    {
        traj_deg(i+ARM_MOTOR_BASE,pose[pose_index][i],FALSE);
    }

    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);

    cout << "Pose No." << pose_index << ": (";
    for(int i=1; i<=ARM_COARSE; i++)
    {
        cout << pose[pose_index][i];
        if(i!=ARM_COARSE)
            cout << ",";
    }
    cout << ")" << endl;

    delay(2000);
if(!read_depth(0, &coord[pose_index], NULL))
{
  /*
   dist=sqrt(coord[pose_index].x*coord[pose_index].x
   +coord[pose_index].y*coord[pose_index].y
   +coord[pose_index].z*coord[pose_index].z);
   dist=9/dist;
   coord[pose_index].x+=dist*coord[pose_index].x;
   coord[pose_index].y+=dist*coord[pose_index].y;
   coord[pose_index].z+=dist*coord[pose_index].z;
   */

  cout << "Depth information read." << endl;
  cout << "Object coordinate: (";
  cout << coord[pose_index].x << ", "
  << coord[pose_index].y << ", "
  << coord[pose_index].z << ")" << endl;
  //Saving learning data
  for(int j=1; j<=ARM_COARSE; j++)
  {
    motordata << pose[pose_index][j];
    motordata << " ";
  }
  motordata << endl;
  objectdata << coord[pose_index].x << " "
  << coord[pose_index].y << " "
  << coord[pose_index].z << " "
  << coord[pose_index].xR << " "
  << coord[pose_index].yR << " "
  << coord[pose_index].xL << " "
  << coord[pose_index].yL << " "
  << endl;
  pose_index++;
  print_motor_pos(ARM_BOARD);
  save_motor_pos(ARM_BOARD, posdata);
  cout << endl;
  posdata << endl;
}
else
{
  cout << "No depth information for this pose." << endl;
  cout << "RANDOMIZING POSE!" << endl;
  randomize_this = 1;
}
num_trials++;

//Returning to starting posture
/*
for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_COARSE; i++)
  m_traj_position(i, 0, 10.0, 0.02, FALSE);
  m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

print_motor_pos(ARM_BOARD);
*/
}

motordata.close();
objectdata.close();
posdata.close();

return_to_start_pos();
print_motor_pos(ARM_BOARD);

return Num_Random_Pose;

int construct_new_mapping()
{
    int pose_index=0;
    int num_trials=0;
    float random_number;
    //float dist;

    srand((unsigned int)(Timestamp::NowAsSeconds ()));

    ofstream motordata("motordata.dat");
    ofstream objectdata("objectdata.dat");
    ofstream posdata("posdata.dat");

    //Ganghua
    //Range of motion in degree.
    long motion_range[NUM_MOTORS+1];
    for(int i=1; i<=NUM_MOTORS; i++)
        motion_range[i]=upperlimit[i]-lowerlimit[i];

    while(pose_index<Num_Random_Pose)
    {
        for(int i=1; i<=ARM_COARSE; i++)
        {
            random_number=(float)rand();

            /*
            if(pose_index==0)
            {
                */
            pose[pose_index][i]=(int)(random_number/RAND_MAX
                *(upperlimit[i]+ARM_MOTOR_BASE]-lowerlimit[i]+ARM_MOTOR_BASE])
                +lowerlimit[i]+ARM_MOTOR_BASE];

            /*
            if(i==3)
                pose[pose_index][i]=-45;
            else if(i==4)
                pose[pose_index][i]=80;
            else if(i==1)
                pose[pose_index][i]=30;
            }
        }
    }
}
{  
  pose[pose_index][i]=pose[0][i];  
}

traj_deg(i+ARM_MOTOR_BASE,pose[pose_index][i],FALSE);

m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

cout << "Pose No." << pose_index << ": (";
for(int i=1; i<=ARM_COARSE; i++)
{
  cout << pose[pose_index][i];
  if(i!=ARM_COARSE)  
    cout << ",";
}
cout << ")" << endl;

delay(1000);

if(!read_depth(0, &coord[pose_index], NULL))
{
  /*
  dist=sqrt(coord[pose_index].x*coord[pose_index].x 
  +coord[pose_index].y*coord[pose_index].y 
  +coord[pose_index].z*coord[pose_index].z);
  dist=9/dist;
  coord[pose_index].x+=dist*coord[pose_index].x;
  coord[pose_index].y+=dist*coord[pose_index].y;
  coord[pose_index].z+=dist*coord[pose_index].z;
  */
  cout << "Depth information read." << endl;
  cout << "Object coordinate: (";
  cout << coord[pose_index].x << ",";
  cout << coord[pose_index].y << ",";
  cout << coord[pose_index].z << ")" << endl;

  //Saving learning data
  for(int j=1; j<=ARM_COARSE; j++)
  {
    motordata << pose[pose_index][j];
    motordata << " ";
  }
  motordata << endl;

  objectdata << coord[pose_index].x << " " 
  << coord[pose_index].y << " " 
  << coord[pose_index].z << " " 
  << coord[pose_index].xR << " " 
  << coord[pose_index].yR << " " 
  << coord[pose_index].xL << " " 
  << coord[pose_index].yL << " " 
  << endl;
pose_index++;

print_motor_pos(ARM_BOARD);
save_motor_pos(ARM_BOARD, posdata);

cout << endl;
posdata << endl;
}
else

cout << "No depth information for this pose." << endl;

num_trials++;

//Returning to starting posture
/*
for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_COARSE; i++)
m_traj_position(i, 0, 10.0, 0.02, FALSE);
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

print_motor_pos(ARM_BOARD);
*/
}

motordata.close();
objectdata.close();
posdata.close();

return_to_start_pos();
print_motor_pos(ARM_BOARD);

return Num_Random_Pose;
}

void statistics_generation(int num_pose)
{
    float motionerror_sum0=0, motionerror_sum1=0;
    Coord3D goal;
    //float dist;

    ofstream statistics("statistics.txt");

    //delete positions of with large error
    ofstream motordata("motordata.dat");
    ofstream objectdata("objectdata.dat");
    int large_errors=0;

    int rep=0;

    for(int i=0; i<num_pose; i++)
    {
        cout << "Pose No." << i << endl;
        statistics << "Pose No." << i << endl;

        cout << "Arm-joint positions: ";
        statistics << "Arm-joint positions: ";
        for(int j=1; j<=ARM_COARSE; j++)
        {
            traj_deg(j+ARM_MOTOR_BASE,pose[i][j],FALSE);
            cout << pose[i][j] << " ";
            statistics << pose[i][j] << " ";
        }
    }
}
cout << endl;
statistics << endl;
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

cout << "Finished." << endl;
statistics << "Finished." << endl;
delay(1000);

if(!read_depth(0, &goal, NULL))
{
    /*
    dist=sqrt(goal.x*goal.x+goal.y*goal.y+goal.z*goal.z);
    dist=9/dist;
    goal.x+=dist*goal.x;
    goal.y+=dist*goal.y;
    goal.z+=dist*goal.z;
    */
    cout << "Goal info: " << coord[i].xR << "," << coord[i].yR << "," << coord[i].xL << "," << coord[i].yL << "," << coord[i].x << "," << coord[i].y << "," << coord[i].z << "," << goal.xR << "," << goal.yR << "," << goal.xL << "," << goal.yL << "," << goal.x << "," << goal.y << "," << goal.z << endl;
    statistics << "Goal info: " << coord[i].xR << "," << coord[i].yR << "," << coord[i].xL << "," << coord[i].yL << "," << coord[i].x << "," << coord[i].y << "," << coord[i].z << "," << goal.xR << "," << goal.yR << "," << goal.xL << "," << goal.yL << "," << goal.x << "," << goal.y << "," << goal.z << endl;

    motion_error[i][0]=(float)((goal.x-coord[i].x)*(goal.x-coord[i].x) + (goal.y-coord[i].y)*(goal.y-coord[i].y) + (goal.z-coord[i].z)*(goal.z-coord[i].z));
    motion_error[i][0]=sqrt(motion_error[i][0]);
    
    motionerror_sum0+=motion_error[i][0];

    /*
    motion_error[i][1]=
    sqrt((float)
    (goal.xR-coord[i].xR)*(goal.xR-coord[i].xR)
    +(goal.yR-coord[i].yR)*(goal.yR-coord[i].yR)
    +(goal.zR-coord[i].zR)*(goal.zR-coord[i].zR));
    motion_error[i][1]=sqrt(motion_error[i][1]);
    */
    motion_error[i][1]=abs(goal.xR-coord[i].xR)+abs(goal.yR-coord[i].yR);
+abs(goal.xL-coord[i].xL)+abs(goal.yL-coord[i].yL);

motionerror_sum1+=motion_error[i][1];

cout << "Error in 3D : " << motion_error[i][0] << " Error in image planes : " <<
motion_error[i][1] << endl;
statistics << "Error in 3D : " << motion_error[i][0] << " Error in image planes : " <<
motion_error[i][1] << endl;

print_motor_pos(ARM_BOARD);
save_motor_pos(ARM_BOARD, statistics);

cout << endl;
statistics << endl;

if(motion_error[i][1]<5)
{
    for(int j=1; j<=ARM_COARSE; j++)
        motordata << pose[i][j] << " ";
    motordata << endl;
    objectdata << coord[i].x << " 
    << coord[i].y << " 
    << coord[i].z << " 
    << coord[i].xR << " 
    << coord[i].yR << " 
    << coord[i].xL << " 
    << coord[i].yL << endl;

    rep=0;
}
else
{
    if(rep==0)
    {
        i--;
        rep=1;
    }
    else
    {
        rep=0;
        large_errors++;
    }
}

} else{
    if(rep==0)
    {
        i--;
        rep=1;
    }
    else
    {
        rep=0;
        large_errors++;
    }
}

cout << "Motion error in 3D (in mm): " << motionerror_sum0 << "/" << motionerror_sum0/num_pose << endl;
cout << "Motion error in image planes (in pixel): " << motionerror_sum1 << "/" <<

}
motionerror_sum1/num_pose << endl;
cout << "Large error: " << large_errors << endl;
statistics << "Motion error in 3D (in mm): " << motionerror_sum0 << "/" << motionerror_sum0/num_pose
<< endl;
statistics << "Motion error in image planes (in pixel): " << motionerror_sum1 << "/" <<
motionerror_sum1/num_pose << endl;
statistics << "Large error: " << large_errors << endl;
statistics.close();
motordata.close();
objectdata.close();

return_to_start_pos();
}
/*
float dist(Coord3D &pos1, Coord3D &pos2)
{
    return(sqrt((pos1.x-pos2.x)*(pos1.x-pos2.x)+(pos1.y-pos2.y)*(pos1.y-pos2.y)+(pos1.z-pos2.z)*(pos1.z-
    pos2.z)));
}
*/
float dist(float *pos1, float *pos2)
{
    pos2[3])));
}

void reaching(Coord3D &goal)
{
    char buffer[10];

cout << endl << "Reading netdata ..." << endl;
ifstream netdata("netdata.dat");
rbfn network(netdata);
netdata.close();
cout << "Reading finished." << endl;

float goal_coord[4];
float motor_coord[ARM_COARSE+1];
float ee_coord[4];
float delta_ee[4];
float delta_mpos[ARM_COARSE+1];

/*** begin: prepare data for coarse reaching ***/
cout << "Goal coordinates: " << goal.x << "," << goal.y << "," << goal.z << endl;
/*
cout << "Goal centroids: (" << goal.xR << "," << goal.yR << ") , 
<< "(" << goal.xL << "," << goal.yL << ")" << endl;
*/
goal_coord[1]=goal.x;
goal_coord[2]=goal.y;
goal_coord[3]=goal.z;

ifstream msp("msp.dat");
for(int i=1; i<=ARM_COARSE; i++)
    msp >> motor_coord[i];
int pcount = 1500;
int degree = 3;
long ** path;
path = new_path_array(pcount + 5, ARM_COARSE);

int b_points = 150;
cout << "pcount = " << pcount << endl;
long ** b_P;
b_P = new long *[ARM_COARSE+1];
b_P = new_path_array(b_points, ARM_COARSE);

network.eval_array(motor_coord, ee_coord);

float distance = dist(ee_coord, goal_coord);
float delta;
int rresult = 0; /* reaching result */

long temp_mpos;
history_counter = 0;

for(int i=0; i<ARM_COARSE; i++)
    b_P[i][0] = 0;
for(int i=3; i<ARM_COARSE; i++)
    b_P[i-1][1] = (long)((float)motor_coord[i]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+i]);

for(int i=0; i<2; i++)
    b_P[i][1] = 0;

for(int i=1; i<=ARM_COARSE; i++)
    b_P[i-1][2] = (long)((float)motor_coord[i]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+i]);

bspline(3, ARM_COARSE, 12, 2, b_P, path);

for (int i = 0; i < 12; i++)
    for (int j = 0; j < ARM_COARSE; j++)
        b_P[j][i] = path[j][i];

history_counter = 12;
cout << endl << "Starting coarse reaching ..." << endl;

while(distance > 200.0) /* 50.0 is the threshold for reaching with visual feedback */
{
    delta = 10.0/distance; /* 10.0 is the step size for reaching */
delta_ee[1]=delta*(goal_coord[1]-ee_coord[1]);
delta_ee[2]=delta*(goal_coord[2]-ee_coord[2]);
delta_ee[3]=delta*(goal_coord[3]-ee_coord[3]);

/*/  
cout << "motor_coord: ";  
for(int i=1; i<=ARM_COARSE; i++)
    cout << motor_coord[i] << " ";
cout << endl;


rresult=network.reaching_one_step_array(motor_coord, delta_ee, delta_mpos);

if (rresult == -1) cout << "Singularity reached!" << endl;

if(rresult <= 0)
{
    cout << "delta_mpos: ";
    for(int i=1; i<=ARM_COARSE; i++)
    {
        if ((lowerlimit[ARM_MOTOR_BASE + i] <= (motor_coord[i] + delta_mpos[i])) &&
            (upperlimit[ARM_MOTOR_BASE + i] >= (motor_coord[i] + delta_mpos[i])))
            {  
                motor_coord[i]+=delta_mpos[i];
                cout << delta_mpos[i] << " ";
            } else {
                cout << " ERR" << i << "] = " << (motor_coord[i] + delta_mpos[i]) << " ";
            }
    }  
    cout << endl;

    cout << "history_counter = " << history_counter << endl;

    if(history_counter<25)
    {
        for(int i=1; i<=ARM_COARSE; i++)
        {
            b_P[i-1][history_counter] = (long)((float)motor_coord[i]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+i]);
            // motor_history[history_counter][i]=motor_coord[i];
            history_counter++;
        } else {
            cout << "Too many steps. Stopping!" << endl;
            break;
        }

    network.eval_array(motor_coord, ee_coord);
    distance=dist(ee_coord, goal_coord);
    }
else
{
    cout << "reaching_one_step failed during coarse reaching." << endl;
    break;
}

}   

// spline(b_points, ARM_COARSE, pcount, b_P, path);
bspline(history_counter, ARM_COARSE, history_counter * 6, degree, b_P, path);
move_on_path(history_counter * 6, path, ARM_COARSE, arm_group, arm_group_addr);

    // cout << endl << "Press <RETURN> to continue with fine reaching." << endl;
    // gets(buffer);

    delete_path_array(path, ARM_COARSE);
    delete_path_array(b_P, ARM_COARSE);

/*
    traj_deg(HEAD_MOTOR_BASE + 4, 0,FALSE);
    m_start_move(head_group_addr, FALSE);
    wait_stop_group(head_group);
*/

delay(2000);

if(!rresult)
{
    Coord3D fine_reaching_start;

    if(!read_depth(2, &fine_reaching_start, &goal))
    {
        // Change to R640480
        ClipInfo clipinfo;
        clipinfo.resolution=2;
        clipinfo.undistort = 1;        //Ganghua originally 1
        clipinfo.format = PIXEL_BGR;

        int x,y;

        x=goal.xR+fine_reaching_start.xR;
        y=goal.yR+fine_reaching_start.yR;

        clipinfo.ctop=x-row_size320/2;
        clipinfo.cleft=y-col_size320/2;

        if(clipinfo.ctop<0)
            clipinfo.ctop=0;
        if(clipinfo.cleft<0)
            clipinfo.cleft=0;

        clipinfo.cbottom=clipinfo.ctop+row_size320;
        clipinfo.cright=clipinfo.cleft+col_size320;

        if(clipinfo.cbottom>row_size640)
            clipinfo.cbottom=row_size640;
        if(clipinfo.cright>col_size640)
            clipinfo.cright=col_size640;

        //cout << clipinfo.ctop << "," << clipinfo.cbottom << "," << clipinfo.cleft << "," << clipinfo.cright << endl;
        /*OutPortClipInfoR.Content()=clipinfo;
        OutPortClipInfoR.Write();*/
        sender_ClipInfoR.Write(&clipinfo);

        x=goal.xL+fine_reaching_start.xL;
y = goal.yL + fine_reaching_start.yL;
clipinfo.ctop = x - row_size320/2;
clipinfo.cleft = y - col_size320/2;

if (clipinfo.ctop < 0)
    clipinfo.ctop = 0;
if (clipinfo.cleft < 0)
    clipinfo.cleft = 0;

clipinfo.cbottom = clipinfo.ctop + row_size320;
clipinfo.cright = clipinfo.cleft + col_size320;

if (clipinfo.cbottom > row_size640)
    clipinfo.cbottom = row_size640;
if (clipinfo.cright > col_size640)
    clipinfo.cright = col_size640;

// cout << clipinfo.ctop << "," << clipinfo.cbottom << "," 
//     << clipinfo.cleft << "," << clipinfo.cright << endl;

/*OutPortClipInfoL.Content() = clipinfo;
OutPortClipInfoL.Write();*/
sender_ClipInfoL.Write(&clipinfo);

cout << endl << "Starting fine reaching ..." << endl << endl;
delay(30000);
    //Ganghua originally 6000

while(!read_depth(2, &fine_reaching_start, &goal))
{
    for (int i = 1; i <= ARM_COARSE; i++)
    {
        m_get_pos(i + ARM_MOTOR_BASE, &temp_mpos, true);
        motor_coord[i] = (float)temp_mpos / 2048 * 360 / gear_ratio[i + ARM_MOTOR_BASE];
    }

    ee_coord[1] = fine_reaching_start.x;
    ee_coord[2] = fine_reaching_start.y;
    ee_coord[3] = fine_reaching_start.z;

    distance = dist(ee_coord, goal_coord);
    if (distance > 10.0)
        delta = 10.0 / distance; /* 10.0 is the step size for reaching */
    else
        delta = 1.0;

    delta_ee[1] = delta * (goal_coord[1] - ee_coord[1]);

    rresult = network.reaching_one_step_array(motor_coord, delta_ee, delta_mpos);
    if (rresult == -1) cout << "Singularity reached!" << endl;
    if (rresult <= 0)
    {
        cout << "delta_mpos: ";
        for (int i = 1; i <= ARM_COARSE; i++)
        {
            cout << "," << motor_coord[i];
            //cout << motor_coord[i] << endl;
        }
    }
motor_coord[i]+=delta_mpos[i];
cout << delta_mpos[i] << " ";
}
cout << endl;

if(history_counter<200)
{
    for(int i=1; i<=ARM_COARSE; i++)
        motor_history[history_counter][i]=motor_coord[i];
    history_counter++;
}

/*
cout << "Press <RETURN> to move the arm ..." << endl;
gets(buffer);
*/

for(int i=1; i<=ARM_COARSE; i++)
    traj_deg(i+ARM_MOTOR_BASE, motor_coord[i],FALSE);
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);
cout << "one fine reaching step finished" << endl;
else
{
    cout << "reaching_one_step failed during fine reaching." << endl;
    break;
}

delay(1000);

/*
cout << "Press <RETURN> to change back to R320240 ..." << endl;
gets(buffer);
*/

// Change back to R320240
clipinfo.resolution=1;
clipinfo.undistort = 1;
clipinfo.format = PIXEL_BGR;
clipinfo.cstop=0; clipinfo.cbottom=row_size320;
clipinfo.cleft=0; clipinfo.cright=col_size320;

/*OutPortClipInfoR.Content()=clipinfo;
OutPortClipInfoR.Write();
OutPortClipInfoL.Content()=clipinfo;
OutPortClipInfoL.Write();*/
sender_ClipInfoR.Write(&clipinfo);
sender_ClipInfoL.Write(&clipinfo);
delay(6000);
else
    cout << "Fine reaching prepared, but only one object observed" << endl;
}
cout << endl << "Press <RETURN> to return the arm the rest position..." << endl;
gets(buffer);

/*
for(int i=history_counter-1; i>=0; i--)
{
    for(int j=1; j<ARM_COARSE; j++)
        traj_deg(j+ARM_MOTOR_BASE, motor_history[i][j], FALSE);
    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);
}
/*

for(int i=1; i<=2; i++)
    m_traj_position(i+ARM_MOTOR_BASE, 0, 10.0, 0.02, FALSE);
    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);

for(int i=3; i<=ARM_COARSE; i++)
    m_traj_position(i+ARM_MOTOR_BASE, 0, 10.0, 0.02, FALSE);
    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);
}

long **reach_and_save(Coord3D & goal, int & size, int & dim)
{
    char buffer[10];

cout << endl << "Reading netdata ..." << endl;
ifstream netdata("netdata.dat");
rbfn network(netdata);
netdata.close();
cout << "Reading finished." << endl;

float goal_coord[4];
float motor_coord[ARM_COARSE+1];
float ee_coord[4];
float delta_ee[4];
float delta_mpos[ARM_COARSE+1];

/*** begin: prepare data for coarse reaching ***/

cout << "Goal coordinates: " << goal.x << "," << goal.y << "," << goal.z << endl;
/*
cout << "Goal centroids: (" << goal.xR << "," << goal.yR << ") , " <<
    "(" << goal.xL << "," << goal.yL << ")" "<< endl;
*/
goal_coord[1]=goal.x;
goal_coord[2]=goal.y;
goal_coord[3]=goal.z;

ifstream msp("msp.dat");
for(int i=1; i<=ARM_COARSE; i++)
    msp >> motor_coord[i];
msp.close();

int pcount = 150;
int degree = 3;

long** path;
path = new_path_array(pcount, ARM_COARSE);

int b_points = 150;
cout << "pcount = " << pcount << endl;

long** b_P;
b_P = new_path_array(b_points, ARM_COARSE);
network.eval_array(motor_coord, ee_coord);

/*** end: prepare data for coarse reaching ***/

float distance=dist(ee_coord, goal_coord);
float delta;
int rresult=0;  /* reaching result */
long temp_mpos;

history_counter=0;

for(int i=0; i<ARM_COARSE; i++)
    b_P[i][0]= 0;

for(int i=3; i<ARM_COARSE; i++)
    b_P[i-1][1] = (long) ((float)motor_coord[i]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+i]);

for(int i=0; i<2; i++)
    b_P[i][1]= 0;

for(int i=1; i<=ARM_COARSE; i++)
    b_P[i-1][2] = (long) ((float)motor_coord[i]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+i]);

bspline(3, ARM_COARSE, 12, 2, b_P, path);

for (int i = 0; i < 12; i++)
    for (int j = 0; j < ARM_COARSE; j++)
        b_P[j][i] = path[j][i];

history_counter = 12;
cout << "Starting coarse reaching ..." << endl << endl;

while(distance>200.0)  /* 50.0 is the threshold for reaching with visual feedback */
{
    delta=10.0/distance;  /* 10.0 is the step size for reaching */

    delta_e[1]=delta*(goal_coord[1]-ee_coord[1]);
    delta_e[2]=delta*(goal_coord[2]-ee_coord[2]);
    delta_e[3]=delta*(goal_coord[3]-ee_coord[3]);

    cout << "motor_coord: ";
    for(int i=1; i<=ARM_COARSE; i++)
        cout << motor_coord[i] << " ";
    cout << endl;

    
    rresult=network.reaching_one_step_array(motor_coord, delta_e, delta_mpos);
for(int mdi = 1; mdi <= ARM_COARSE; mdi++)
  delta_mpos[mdi] *= 3.0;

if (rresult == -1) cout << "Singularity reached!" << endl;
if(rresult <= 0)
{
  cout << "delta_mpos: ";
  for(int i=1; i<=ARM_COARSE; i++)
  {
    if (lowerlimit[ARM_MOTOR_BASE + i] > (motor_coord[i] + delta_mpos[i])) {
      motor_coord[i] = lowerlimit[ARM_MOTOR_BASE + i];
      cout << " LL" << i << " = " << (motor_coord[i] + delta_mpos[i]) << " ";
    } else if (upperlimit[ARM_MOTOR_BASE + i] < (motor_coord[i] + delta_mpos[i])) {
      motor_coord[i] = upperlimit[ARM_MOTOR_BASE + i];
      cout << " UL" << i << " = " << (motor_coord[i] + delta_mpos[i]) << " ";
    } else {
      motor_coord[i] += delta_mpos[i];
      cout << delta_mpos[i] << " ";
    }
  }
  cout << endl;
  cout << "history_counter = " << history_counter << endl;
  if(history_counter<25)
  {
    for(int i=1; i<=ARM_COARSE; i++)
      b_P[i-1][history_counter] = (long)((float)motor_coord[i]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+i]);
    //  motor_history[history_counter][i]=motor_coord[i];
    history_counter++;
  } else {
    cout << "Too many steps. Stopping!" << endl;
    break;
  }
  network.eval_array(motor_coord, ee_coord);
  distance=dist(ee_coord, goal_coord);
}
else
{
  cout << "reaching_one_step failed during coarse reaching." << endl;
  break;
}

// spline(b_points, ARM_COARSE, pcount, b_P, path);
// bspline(history_counter, ARM_COARSE, history_counter * 6, degree, b_P, path);
// move_on_path(history_counter * 6, path, ARM_COARSE, arm_group, arm_group_addr);
// cout << endl << "Press <RETURN> to continue with fine reaching." << endl;
// gets(buffer);

delay(2000);
if(!result)
{
    Coord3D fine_reaching_start;

    // if(!read_depth(2, &fine_reaching_start, &goal)) || (delay(1000)) || (!read_depth(0, &fine_reaching_start, &goal))
    if(!read_depth(2, &fine_reaching_start, &goal))
    {
        // Change to R640480
        ClipInfo clipinfo;
        clipinfo.resolution=2;
        clipinfo.undistort = 1;          // Ganghua
        clipinfo.format = PIXEL_BGR;

        int x,y;

        x=goal.xR+fine_reaching_start.xR;
        y=goal.yR+fine_reaching_start.yR;

        clipinfo.ctop=x-row_size320/2;
        clipinfo.cleft=y-col_size320/2;

        if(clipinfo.ctop<0)
            clipinfo.ctop=0;
        if(clipinfo.cleft<0)
            clipinfo.cleft=0;

        clipinfo.cbottom=clipinfo.ctop+row_size320;
        clipinfo.cright=clipinfo.cleft+col_size320;

        if(clipinfo.cbottom>row_size640)
            clipinfo.cbottom=row_size640;
        if(clipinfo.cright>col_size640)
            clipinfo.cright=col_size640;

        //cout << clipinfo.ctop << "," << clipinfo.cbottom << "," << cout
        // << clipinfo.cleft << "," << clipinfo.cright << endl;

        /*OutPortClipInfoR.Content()=clipinfo;
        OutPortClipInfoR.Write();*/
        sender_ClipInfoR.Write(&clipinfo);

        x=goal.xL+fine_reaching_start.xL;
        y=goal.yL+fine_reaching_start.yL;

        clipinfo.ctop=x-row_size320/2;
        clipinfo.cleft=y-col_size320/2;

        if(clipinfo.ctop<0)
            clipinfo.ctop=0;
        if(clipinfo.cleft<0)
            clipinfo.cleft=0;

        clipinfo.cbottom=clipinfo.ctop+row_size320;
        clipinfo.cright=clipinfo.cleft+col_size320;

        if(clipinfo.cbottom>row_size640)
            clipinfo.cbottom=row_size640;
        if(clipinfo.cright>col_size640)
            clipinfo.cright=col_size640;
clipinfo.cright=col_size640;

//cout << clipinfo.ctop << "," << clipinfo.cbottom << ","  
// " << clipinfo.cleft << "," << clipinfo.cright << endl;

/*OutPortClipInfoL.Content()=clipinfo;  
OutPortClipInfoL.Write();*/
sender_ClipInfoL.Write(&clipinfo);

cout << endl << "Starting fine reaching ..." << endl << endl;
delay(6000);  
//Ganghua originally 6000

int md_continue = 1;

// if(!(read_depth(2, &fine_reaching_start, &goal)) || (delay(1000)) || (read_depth(0,  
&fine_reaching_start, &goal)))
while((md_continue) && (read_depth(2, &fine_reaching_start, &goal)))
   while (!(read_depth(2, &fine_reaching_start, &goal)) || (delay(300)) || (read_depth(0,  
&fine_reaching_start, &goal)))
{
   cout << "obj: " << goal.x << ", " << goal.y << ", " << goal.z << endl;
   cout << "arm: " << fine_reaching_start.x << ", " << fine_reaching_start.y << ", " << fine_reaching_start.z << endl;
for(int i=1; i<=ARM_COARSE; i++)
   {
      m_get_pos(i+ARM_MOTOR_BASE, &temp_mpos,true);
      motor_coord[i] = (float)temp_mpos/2048*360/gear_ratio[i+ARM_MOTOR_BASE];
   }

ee_coord[1]=fine_reaching_start.x;  
ee_coord[2]=fine_reaching_start.y;  
ee_coord[3]=fine_reaching_start.z;

distance=dist(ee_coord, goal_coord);

if(distance>10.0)
   delta=10.0/distance; /* 10.0 is the step size for reaching */
else
   delta=1.0;

   delta_ee[1]=delta*(goal_coord[1]-ee_coord[1]);
   delta_ee[2]=delta*(goal_coord[2]-ee_coord[2]);
   delta_ee[3]=delta*(goal_coord[3]-ee_coord[3]);

rresult=network.reaching_one_step_array(motor_coord, delta_ee, delta_mpos);

//  
for(int mdi = 1; mdi <= ARM_COARSE; mdi++)
   
 delta_mpos[mdi] *= 3.0;

if (rresult == -1) cout << "Singularity reached!" << endl;
if(rresult == 0)
{
   cout << "delta_mpos: ";
   for(int i=1; i<=ARM_COARSE; i++)
      {
if (lowerlimit[ARM_MOTOR_BASE + i] > (motor_coord[i] +
  delta_mpos[i])) {
    motor_coord[i] = lowerlimit[ARM_MOTOR_BASE + i];
    cout << " LL[" << i << "] = " << (motor_coord[i] +
  delta_mpos[i]) << " ";
} else if (upperlimit[ARM_MOTOR_BASE + i] < (motor_coord[i] +
  delta_mpos[i])) {
    motor_coord[i] = upperlimit[ARM_MOTOR_BASE + i];
    cout << " UL[" << i << "] = " << (motor_coord[i] +
  delta_mpos[i]) << " ";
} else {
    motor_coord[i]+=delta_mpos[i];
    cout << delta_mpos[i] << " ";
    //
    motor_coord[i]+=delta_mpos[i];
    //
    cout << delta_mpos[i] << " ";
  }
cout << endl;

if(history_counter< 50)
{
  for(int i=1; i<=ARM_COARSE; i++)
    b_P[i­1][history_counter] = (long)
((float)motor_coord[i]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+i]);
  //
  motor_history[history_counter][i]=motor_coord[i];
  history_counter++;
}
/*
cout << "Press <RETURN> to move the arm ..." << endl;
gets(buffer);
*/

for(int i=1; i<=ARM_COARSE; i++)
  traj_deg(i+ARM_MOTOR_BASE, motor_coord[i],FALSE);
  m_start_move(arm_group_addr, FALSE);
  wait_stop_group(arm_group);
  cout << "one fine reaching step finished" << endl;
}
else
{
  cout << "reaching_one_step failed during fine reaching." << endl;
  break;
}

md_continue­­;
if (md_continue) {
  delay(1000);  // Connor old value 1000
} else {
  cout << "; ";
  gets(buffer);
  sscanf(buffer, "%d", &md_continue);
}
/*
cout << "Press <RETURN> to change back to R320240 ... " << endl;
gets(buffer);
*/
// Change back to R320240
clipinfo.resolution = 1;
clipinfo.undistort = 1;
clipinfo.format = PIXEL_BGR;
clipinfo.ctop = 0; clipinfo.cbottom = row_size320;
clipinfo.cleft = 0; clipinfo.cright = col_size320;

/* OutPortClipInfoR.Content() = clipinfo; 
OutPortClipInfoR.Write(); 
OutPortClipInfoL.Content() = clipinfo; 
OutPortClipInfoL.Write(); */
sender_ClipInfoR.Write(&clipinfo);
sender_ClipInfoL.Write(&clipinfo);
delay(6000);

} 
else 

cout << "Fine reaching prepared, but only one object observed" << endl;
}
cout << endl << "Press <RETURN> to return the arm the rest position..." << endl;
gets(buffer);

for (int i = 1; i <= 2; i++)
    m_traj_position(i+ARM_MOTOR_BASE, 0, 10.0, 0.02, FALSE);
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

for (int i = 3; i <= ARM_COARSE; i++)
    m_traj_position(i+ARM_MOTOR_BASE, 0, 10.0, 0.02, FALSE);
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

long** ret_path;
ret_path = new_path_array(history_counter, ARM_COARSE);

dim = ARM_COARSE;
size = history_counter;

for (int i = 0; i < history_counter; i++)
    for (int j = 0; j < ARM_COARSE; j++) {
        ret_path[j][i] = b_P[j][i];
    }
delete_path_array(path, ARM_COARSE);
delete_path_array(b_P, ARM_COARSE);
return ret_path;

}

void kine_learning_fast_without_user_input()
{
    cout << "Start inverse kinematics learning in fast-mode..." << endl;

    int num_pose = 0;
    num_pose = construct_new_mapping();

    statistics_generation(num_pose);
//Copy motordata.dat and objectdata.dat to peacock
system("on -d -u marek -f kronos /home/marek/bin/matlab1");

return_to_start_pos();

cout << endl;
cout << endl;
cout << "Press <RETURN> to return to weird mode ...";
char buffer[10];
gets(buffer);
cout << endl;
cout << endl;
cout << "KINEMATICS LEARNING IS DONE. YOU CAN RESTART THE SYSTEM NOW!" << endl;
cout << endl;
cout << endl;
}

void kine_learning()
{

cout << "Start inverse kinematics learning..." << endl;
char buffer[10];

int num_pose=0;
buffer[0]='!';
while(buffer[0]!='r' && buffer[0]!='c')
{
    cout << "Press r to Read kinematics mapping." << endl;
    cout << "Or press c to Construct new mapping." << endl;
    cout << "Or press l to Construct new mapping using old motordata." << endl;
    flushall();
    gets(buffer);

    if(buffer[0]=='r')
        num_pose=get_kine_mapping();
    else if(buffer[0]=='c')
        num_pose=construct_new_mapping();
    else if(buffer[0]=='l')
        num_pose=construct_new_mapping_old();
}

buffer[0]='!';
while(buffer[0]!='g' && buffer[0]!='s' )
{
    cout << "Press g to Generate statistics of saved poses..." << endl;
    cout << "Press s to Skip this step..." << endl;
    flushall();
    gets(buffer);

    if(buffer[0]=='g')
        statistics_generation(num_pose);
    else if(buffer[0]=='s')
        break;
}

//Copy motordata.dat and objectdata.dat to peacock
system("on -d -u marek -f kronos /home/marek/bin/matlab1");

cout << "Press <RETURN> to start testing ... ";
gets(buffer);
Coord3D goal;
// marek: float arm_coord[ARM_COARSE+1];
//float dist;
//int vianum;

while(1)
{
    buffer[0]='!';
    while(buffer[0]!='v' && buffer[0]!='m' && buffer[0]!='e')
    {
        cout << endl;
        cout << "Press v to move to a visually detectable object position..." << endl;
        cout << "Press m to manually specify arm-joint positions..." << endl;
        cout << "Press e to Exit this program..." << endl;
        flushall();
        gets(buffer);
    }
    if(buffer[0]=='e')
        break;
    if(buffer[0]=='v')
    {
        if(!read_depth(1, NULL, &goal))
        {
            reaching(goal);
        }
        else
        {
            cout << "No object found." << endl;
        }
    }
    else
    {
        /*
        for(int i=1; i<=ARM_COARSE; i++)
        {
            cout << "Coordinate for arm motor No. " << i << ": ";
            flushall();
            gets(buffer);
            arm_coord[i]=atoi(buffer);
            traj_deg(i+ARM_MOTOR_BASE,(int)arm_coord[i],FALSE);
        }
        m_start_move(arm_group_addr, FALSE);
        wait_stop_group(arm_group);
        delay(2000);
        */
        if(!read_depth(0, &goal, NULL))
        {
            dist=sqrt(goal.x*goal.x+goal.y*goal.y+goal.z*goal.z);
            dist=9/dist;
            goal.x=dist*goal.x;
            goal.y=dist*goal.y;
            goal.z=dist*goal.z;
            xx
            cout << "Goal coordinates: " goal.x << "," goal.y << "," goal.z << endl;
            cout << "Goal centroids: (" goal.xR << "," goal.yR << ") , "
        }
    }
}
cout << "Press <RETURN> to return to start position." << endl;
gets(buffer);
return_to_start_pos();
}
*/

}/*
cout << "Press <RETURN> to continue...";
gets(buffer);
return_to_start_pos();
*/
if(history_counter!=0)
{
    cout << "Step start" << endl;
    for(int i=0; i<=history_counter­1; i++)
    {
        for(int j=1; j<=ARM_COARSE; j++)
            traj_deg(j+ARM_MOTOR_BASE, motor_history[i][j], FALSE);
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);
if(i==0)
    print_motor_pos(ARM_BOARD);
    }
cout << endl << "Press <RETURN> to return to start position." << endl;
gets(buffer);
print_motor_pos(ARM_BOARD);
return_to_start_pos();
cout << "Continuous start" << endl;

long path[ARM_COARSE+1][2000];
long** path;
path = new_path_array(2000, ARM_COARSE);
long** b_P;
b_P = new_path_array(history_counter + 30, ARM_COARSE);

cout << "history_counter = " << history_counter << endl;

ofstream path_file("path.dat");

path_file << (history_counter + 3) << " ";

for (int k=1; k<=ARM_COARSE; k++) {
    b_P[k-1][0] = 0;
    path_file << b_P[k-1][1] << " ";
}

for (int k=1; k<=ARM_COARSE; k++) {
    if (k > 2) {
        b_P[k-1][1] = (long)
                    ((float)motor_coord[k]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+k]);
    } else {
        b_P[k-1][1] = 0;
    }
    path_file << b_P[k-1][1] << " ";
}

for (int k=1; k<=ARM_COARSE; k++) {
    b_P[k-1][2] = (long)
                ((float)motor_coord[k]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+k]);
    path_file << b_P[k-1][2] << " ";
}

for(int i=0; i<history_counter; i++)
{
    for(int k=1; k<=ARM_COARSE; k++)
    {
        b_P[k-1][i+3] = (long)
                        ((float)motor_history[i][k]*2048.0/360.0*(float)gear_ratio[ARM_MOTOR_BASE+k]);
    }
    path_file << b_P[i+3] << " ";
}

path_file << endl;

//

spline(history_counter + 3, ARM_COARSE, history_counter * 20, b_P, path);
bspline(history_counter + 3, ARM_COARSE, history_counter * 20, 3, b_P, path);
move_on_path(history_counter * 20, path, ARM_COARSE, arm_group, arm_group_addr);

cout << endl << "Press <RETURN> to return to start position." << endl;
gets(buffer);
print_motor_pos(ARM_BOARD);
return_to_start_pos();

for (int i = 0; i < ARM_COARSE+1; i++)
    delete path[i];
delete path;

for (int i = 0; i < ARM_COARSE+1; i++)
    delete b_P[i];
delete b_P;
}
else
{
    long path[7];
long last_path_point=0;
float acceleration=40.0;  // deg/sec/sec
float velocity_max=40.0;  // deg/sec
float velocity=0.0;
int path_points=0;
int path_segment_points=0;
byte npoints;

cout << "Before starting." << endl;
print_motor_pos(ARM_BOARD);

while(path_points<88)
{
    m_get_npoints(9, &npoints, true);
    cout << "npoints = " << (int)npoints << endl;

    path_segment_points=0;
    while(path_segment_points<7)
    {
        if(path_points<60)
        {
            if(velocity<velocity_max)
                velocity+=acceleration/30.0;
        }
        else
        {
            if(velocity>0)
                velocity-=acceleration/30.0;
        }

        path[path_segment_points] = (long)(velocity/30.0*2048/360*gear_ratio[9]+last_path_point);
        cout << path[path_segment_points] << " " ;

        last_path_point=path[path_segment_points];
        path_segment_points++; path_points++;
    }
    cout << endl;

    m_add_path_points(9,7,path);
delay(100);
}

path[0]=last_path_point;
m_add_path_points(9,1,path);
delay(100);

m_get_npoints(9, &npoints, true);
cout << "npoints = " << (int)npoints << endl;

m_start_path(arm_group_addr, 9);
delay(8000);

cout << "After path." << endl;
print_motor_pos(ARM_BOARD);

return_to_start_pos();
cout << "After returning." << endl;  
print_motor_pos(ARM_BOARD);

return_to_start_pos();

// make_moves();

void display_wait(int d)
{
    cout << "WAIT " << d << " sec: ";
    for (int i = 0; i < d; i++)
    {
        cout << (i % 10);
        cout.flush();
        delay(1000);
    }
    cout << endl;
}

// Eric Inserted this

int HeadReinforcement(int typeR, int iterations)
{//this nods according to how well you do, shakes if you do poorly

    traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);

    switch (typeR)
    {
    case -1: cout << "Shake head NO." << endl; //you did badly
        //small change... 10 to iterations
        for (int temp_i = 0; temp_i < iterations; temp_i++)
        {
            traj_deg(HEAD_MOTOR_BASE + NECK_YAW, (float)((40 * (temp_i % 2)) - 20), FALSE);
            m_start_move(head_group_addr, FALSE);
            wait_stop_group(head_group);
        }
        traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
        m_start_move(head_group_addr, FALSE);
        wait_stop_group(head_group);
        break;

    case 1: cout << "Nod head YES." << endl; //you did well
        //small change... 10 to iterations
        for (int temp_i = 0; temp_i < iterations; temp_i++)
        {
            traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, (float)((30 * (temp_i % 2)) - 15), FALSE);
        }
    }
m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);
}
traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);

break;
case 0: cout << "No feedback." << endl; //you remained neutral
    break;
default : cout << "Wrong args to Head Reinforcement\n";
    return 0; //fail
}
return 1; //success

int make_moves()
{
    int pose_index=0;
    int num_trials=0;
    //****************************
    // Erica Inserted this    *
    //****************************
    int typeR = -1; //type of reinforcement
    int iterations = 10; //of each behavior
    //int goToFunction = 1; // will ensure i bypass while loop
    int processLoop = 1; //*** will keep while loop running
    //****************************
    cout << "Welcome to Nico's weird-mode!" << endl;
    cout << "Type your command or ? for help. " << endl;

    //****************************
    //while(1) became while (processLoop == 1), initialized to 1
    //changed only in the elseif that runs headReinforcement
    //****************************
    while(processLoop == 1)
    {
        cout << "in while\n";
        cout.flush();
        cout << ":?";
        char buffer[100];
        char keybuffer[100];
        gets(buffer);
        char *pathsfn[10];
        char pfn1[200]; pfn1[0] = 0;
        char pfn2[200]; pfn2[0] = 0;
        char pfn3[200]; pfn3[0] = 0;
char pfn[4][200]; pfn[4][0] = 0;
char pfn[5][200]; pfn[5][0] = 0;
char pfn[6][200]; pfn[6][0] = 0;
char pfn[7][200]; pfn[7][0] = 0;
char pfn[8][200]; pfn[8][0] = 0;
char pfn[9][200]; pfn[9][0] = 0;
char pfn[10][200]; pfn[10][0] = 0;

pathsfn[0] = pfn1;
pathsfn[1] = pfn2;
pathsfn[2] = pfn3;
pathsfn[3] = pfn4;
pathsfn[4] = pfn5;
pathsfn[5] = pfn6;
pathsfn[6] = pfn7;
pathsfn[7] = pfn8;
pathsfn[8] = pfn9;
pathsfn[9] = pfn10;

char tmp[1][200], tmp2[200], tmpL[200], tmpR[200], shtmp[200];

int parm1 = 1;
int parm2 = 1;

int pcount = 200; // slow: 300; fast: 180;
int degree = 3;

if (buffer[0]=='?') { //****************************
    // Erica Inserted this    *
    //****************************
cout << endl;
cout << "Weird mode commands:" << endl;
cout << "a i = x   : move arm motor i to position x. " << endl;
cout << "h i = x   : move head motor i to position x. " << endl;
cout << "l [n]     : load motordata.dat and objectdata.dat." << endl;
    cout << "    If n is specified, load only n first lines. " << endl;
cout << "p i       : Move to pose i and output debug info. " << endl;
cout << "y a b c   : Interpolation demo with degree a and b" << endl;
cout << "    points to be interpolated. " << endl;
cout << "    if c = 1 bezier interpolation is used," << endl;
cout << "    otherwise bsplines are used. " << endl;
cout << "k         : Start kine learning mode. " << endl;
cout << "q         : Let Nico rest for a while... " << endl;
cout << "?         : Display this help page. " << endl;
}

cout << endl;
}
cout.flush();
}
cout << "Completed HeadIterations\n";
//processLoop = 0; //*** Changed while(1) to a var
cout.flush();
}
else if (sscanf(buffer, "a %d = %d", &parm1, &parm2))
{
    cout << "Move arm motor " << parm1 << " to " << parm2 << "." << endl;
    //
    pose[pose_index][parm1] = parm2;
    //
    traj_deg(parm1 + ARM_MOTOR_BASE,pose[pose_index][parm1], FALSE);
    traj_deg(parm1 + ARM_MOTOR_BASE, parm2, FALSE);
    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);
}
else if (sscanf(buffer, "h %d = %d", &parm1, &parm2))
{
    cout << "Move head motor " << parm1 << " to " << parm2 << "." << endl;
    traj_deg(parm1 + HEAD_MOTOR_BASE, parm2, FALSE);
    m_start_move(head_group_addr, FALSE);
    wait_stop_group(head_group);
}
else if (!strcmp(buffer, "arm down"))
{
    cout << "Move arm to rest position." << endl;
    traj_deg(ARM_MOTOR_BASE + 1, 0, FALSE);
    traj_deg(ARM_MOTOR_BASE + 2, 0, FALSE);
    traj_deg(ARM_MOTOR_BASE + 3, 0, FALSE);
    traj_deg(ARM_MOTOR_BASE + 4, 0, FALSE);
    traj_deg(ARM_MOTOR_BASE + 5, 0, FALSE);
    traj_deg(ARM_MOTOR_BASE + 6, 0, FALSE);
    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);
}
else if (!strcmp(buffer, "finger"))
{
    cout << "Show the finger!" << endl;
    traj_deg(ARM_MOTOR_BASE + 1, 40, FALSE);
    traj_deg(ARM_MOTOR_BASE + 2, 0, FALSE);
    traj_deg(ARM_MOTOR_BASE + 3, 0, FALSE);
    traj_deg(ARM_MOTOR_BASE + 4, 70, FALSE);
    traj_deg(ARM_MOTOR_BASE + 5, 20, FALSE);
    traj_deg(ARM_MOTOR_BASE + 6, 100, FALSE);
    m_start_move(arm_group_addr, FALSE);
    wait_stop_group(arm_group);
}
else if (!strcmp(buffer, "head up"))
{
    cout << "Move head to upright position." << endl;
    traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 35, FALSE);
    traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);
    m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);

} else if (!strcmp(buffer, "head shake no"))
{
    cout << "Shake head." << endl;
    traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);
    for (int temp_i = 0; temp_i < 10; temp_i++)
    {
        traj_deg(HEAD_MOTOR_BASE + NECK_YAW, (float)((40 * (temp_i % 2)) - 20), FALSE);
        m_start_move(head_group_addr, FALSE);
        wait_stop_group(head_group);
    }
    traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
    m_start_move(head_group_addr, FALSE);
    wait_stop_group(head_group);
}
else if (!strcmp(buffer, "head shake yes"))
{
    cout << "Shake head." << endl;
    traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);
    for (int temp_i = 0; temp_i < 10; temp_i++)
    {
        traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, (float)((30 * (temp_i % 2)) - 15), FALSE);
        m_start_move(head_group_addr, FALSE);
        wait_stop_group(head_group);
    }
    traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
    m_start_move(head_group_addr, FALSE);
    wait_stop_group(head_group);
}
else if (!strcmp(buffer, "head down"))
{
    cout << "Move head to lowered position." << endl;
    traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 0, FALSE);
    traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);
    m_start_move(head_group_addr, FALSE);
    wait_stop_group(head_group);
}
else if (!strcmp(buffer, "head startpos"))
{
    return_head_to_start_pos();
}
else if (!strcmp(buffer, "reset"))

{
    //Reset motor positions
    for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
        m_reset_pos(i, false);

    for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
        m_reset_pos(i, false);

    cout << "All motor positions reset to 0." << endl;
}
else if (sscanf(buffer, "saveopos %s", filename))
{
    if (filename[0] == 0)
    {
        cout << "correct usage: saveopos filename." << endl;
    }
    else
    {
        //sprintf(tmp1, "%s%s", PATHS_PATH, filename);
        sprintf(tmp2, "%s%s.opos", PATHS_PATH, filename);

        Coord3D goal;

        if(!read_depth(1, NULL, &goal))
        {
            cout << "TOPLEVEL: Save object position." << endl;
            save_object_pos(tmp2, goal);
        }
        else
            cout << "No object found." << endl;
    }
}
else if (sscanf(buffer, "setvar %d %d", &parm1, &parm2))
{
    global_config[parm1] = parm2;
    cout << "Global configuration variable #" << parm1 << " set to " << parm2 << "." << endl;
}
else if (sscanf(buffer, "reach %s %d", filename, &parm1))
{
    if (filename[0] == 0)
    {
        cout << "correct usage: reach filename." << endl;
    }
    else
    {
        //sprintf(tmp1, "%s%s", PATHS_PATH, filename);
        sprintf(tmp2, "%s%s.opos", PATHS_PATH, filename);

        cout << "TOPLEVEL: Move head down." << endl;

        traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
        traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
        traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 0, FALSE);
        traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);
    }
}
m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);
delay(2000);
cout << "TOLEVEL: Reach." << endl;

// Copy motordata.dat and objectdata.dat to peacock
system("on -d -u marek -f kronos /home/marek/bin/matlab1");
get_kine_mapping();
Coord3D goal;
if(!read_depth(1, NULL, &goal))
{
cout << "TOLEVEL: Save object position." << endl;
save_object_pos(tmp2, goal);
long **rpath;
int dim, size;
rpath = reach_and_save(goal, size, dim);
cout << "TOLEVEL: Save path." << endl;
save_path_to_file(tmp1, rpath, size, dim);
cout << "TOLEVEL: Save path." << endl;
delete_path_array(rpath, dim);
/*
long **lpath;
lpath = load_path_from_file(tmp1, size, dim);
cout << "TOLEVEL: loadpath, size = " << size << " , dim = " << dim << endl;
long **mpath = new_path_array(180, dim);
bspline(size, dim, 180, 3, lpath, mpath);
cout << "TOLEVEL: Start to move." << endl;
move_on_path(180, mpath, dim, arm_group, arm_group_addr);
cout << endl << "Press <RETURN> to return to start position." << endl;
gets(keybuffer);
print_motor_pos(ARM_BOARD);
return_to_start_pos();
delete_path_array(mpath, dim);
delete_path_array(lpath, dim);
*/
}
else

cout << "No object found." << endl;
}

else if (0 < sscanf(buffer, "play %s %d %d", filename, &degree, &pcount))
if (filename[0] == 0) {
    cout << "correct usage: play filename [degree [pcount]]." << endl;
} else {
    sprintf(tmp1, "%s%s", PATHS_PATH, filename);
    int dim, size;
    long **lpath;
    lpath = load_path_from_file(tmp1, size, dim);
    cout << "TOPLEVEL: loadpath, size = " << size << " , dim = " << dim << endl;
    long **mpath = new_path_array(pcount, dim);
    bspline(size, dim, pcount, degree, lpath, mpath);
    cout << "TOPLEVEL: Start to move." << endl;
    move_on_path(pcount, mpath, dim, arm_group, arm_group_addr);
    cout << endl << "Press <RETURN> to return to start position." << endl;
    gets(keybuffer);
    print_motor_pos(ARM_BOARD);
    return_to_start_pos();
    delete_path_array(mpath, dim);
    delete_path_array(lpath, dim);
}

} else if (sscanf(buffer, "pointrec %s %s %s %s %s %s %s %s %s %s %s %d %d", filename, pfn1, pfn2, pfn3, pfn4, pfn5, pfn6, pfn7, pfn8, pfn9, &degree, &pcount)) {
    if ((pfn1[0] == 0) || (filename[0] == 0)) {
        cout << "correct usage: pointrec videofilename pathfn1 [pathfn2 [ ... [pathfn5] ... ]."
        << endl; // [degree [pcount]]." << endl;
    } else {
        int pfnc = 1;
        while ((pfnc < 10) && (pathsfn[pfnc][0] != 0))
            pfnc++;
        cout << "Number of paths: " << pfnc << endl;
        sprintf(tmpL, "%s%s/%sL", VIDEO_PATH, filename, filename);
        sprintf(tmpR, "%s%s/%sR", VIDEO_PATH, filename, filename);
        sprintf(shtmp, "/home/marek/video/%s.sh", filename);
        cout << "Video files: " << tmpL << ", " << tmpR << endl;
        cout << "Shell script file: " << shtmp << endl;

        // START PREPARING DIRECTORIES AND SHELL SCRIPTS
char exec_cmd[200];
sprintf(exec_cmd, "mkdir /net/node5%s%s; mkdir /net/node6%s%s", VIDEO_PATH, filename, VIDEO_PATH, filename);

cout << "make paths:" << endl;
cout << exec_cmd << endl;
system(exec_cmd);

// FILE PROCESSING (tar) STARTS HERE

ofstream shfile(shtmp);

for (int i = 0; i < pfnc; i++)
{
    sprintf(exec_cmd, "cp %s%s opos /home/marek/video/%s.%d.path", PATHS_PATH, pathsfn[i], filename, i);
    shfile << exec_cmd << endl;
    // sprintf(exec_cmd, "cp %s%s.opos /home/marek/video/%s.%d.opos", PATHS_PATH, pathsfn[i], filename, i);
    if (i == 0) {
        sprintf(exec_cmd, "cat %s%s.opos > /home/marek/video/%s.opos", PATHS_PATH, pathsfn[i], filename);
    } else {
        sprintf(exec_cmd, "cat %s%s.opos >> /home/marek/video/%s.opos", PATHS_PATH, pathsfn[i], filename);
    }
    shfile << exec_cmd << endl;
}

sprintf(exec_cmd, "cp /home/marek/video/%s.opos /net/node5%s%s", filename, VIDEO_PATH, filename);
shfile << exec_cmd << endl;
sprintf(exec_cmd, "on -u root -f node5 sh -c "cd /net/node5%s%s; tar zcf %sR.tar.gz *");
shfile << exec_cmd << endl;
sprintf(exec_cmd, "on -u root -f node6 sh -c "cd /net/node6%s%s; tar zcf %sL.tar.gz *");
shfile << exec_cmd << endl;

 Coord3D goal;
// load_object_pos(tmp2, goal);
cout << "Object position: X= " << goal.x << " Y= " << goal.y << " Z= " << goal.z << endl;

cout << "Lower head to imitate looking at object." << endl;

traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);

m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);

delay(1000);

cout << "Move head back to upright position." << endl;

traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 35, FALSE);
traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);

m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);

// START RECORDING HERE

// Change back to R320240
ClipInfo clipinfo;
clipinfo.resolution=1;
clipinfo.undistort = 0;
clipinfo.format = 1; // PIXEL_MONO;
clipinfo.ctop=0; clipinfo.cbottom=row_size320;
clipinfo.cleft=0; clipinfo.cright=col_size320;

/*OutPortClipInfoR.Content()=clipinfo;
OutPortClipInfoR.Write();
OutPortClipInfoL.Content()=clipinfo;
OutPortClipInfoL.Write();*/
sender_ClipInfoR.Write(&clipinfo);
sender_ClipInfoL.Write(&clipinfo);

delay(3000);

RecCommand reccommand;
reccommand.recording = RECORDING_START;
reccommand.frameNum = VIDEO_FRAMES;

long tstart = time_msec() + 5000L;
reccommand.start = tstart;

cout << "Videolength will be " << VIDEO_FRAMES << " frames." << endl;
cout << "Sending RECORDING_START with startime = " << tstart << ". Current time is " << time_msec() << "." << endl;

strcpy(recommand.filePrefix, tmpR);
/*OutPortRecCommandR.Content() = reccommand;
OutPortRecCommandR.Write();*/
sender_RecCommandR.Write(&recommand);
strcpy(recommand.filePrefix, tmpL);
cout << "TOPLEVEL: Starting to record videos NOW!" << endl;

display_wait(45);

int dim, size;
long **lpath;

for (int reps = 1; reps <= pfnc; reps++)
{
    sprintf(tmp1, "%s%s", PATHS_PATH, pathsfn[reps - 1]);
    sprintf(tmp2, "%s%s.opos", PATHS_PATH, pathsfn[reps - 1]);
    cout << "Path file for object # " << reps << " : " << tmp1 << endl;
    cout << "Object position file : " << tmp2 << endl;

    lpath = load_path_from_file(tmp1, size, dim);
    cout << "TOPLEVEL: loadpath, size = " << size << " , dim = " << dim << endl;

    long **mpath = new_path_array(pcount, dim);
    bspline(size, dim, pcount, degree, lpath, mpath);
    cout << "Video time = " << (time_msec() - tstart) << endl;
    cout << "TOPLEVEL: Start to move." << endl;

    reccommand.recording = RECORDING_MARK;
    reccommand.marker = 10 * reps;
    /*OutPortRecCommandR.Content() = reccommand;
    OutPortRecCommandR.Write();*/
    OutPortRecCommandL.Content() = reccommand;
    OutPortRecCommandL.Write(&reccommand);
}

//
move_on_path(pcount, mpath, dim, arm_group, arm_group_addr);
//
wait_stop_group(arm_group);

/*/ cout << endl << "Press <RETURN> to return to start position." << endl;
gets(keybuffer);
*/

reccommand.recording = RECORDING_MARK;
reccommand.marker = 10 * reps + 1;
/*OutPortRecCommandR.Content() = reccommand;
OutPortRecCommandR.Write();*/
OutPortRecCommandL.Content() = reccommand;
OutPortRecCommandL.Write(&reccommand);
sender_RecCommandR.Write(&reccommand);
sender_RecCommandL.Write(&reccommand);
cout << "Video time = " << (time_msec() - tstart) << endl;

display_wait(2);

reccommand.recording = RECORDING_MARK;
reccommand.marker = 10 * reps + 2;
/*OutPortRecCommandR.Content() = reccommand;
OutPortRecCommandR.Write();
OutPortRecCommandL.Content() = reccommand;
OutPortRecCommandL.Write();*/
sender_RecCommandR.Write(&reccommand);
sender_RecCommandL.Write(&reccommand);

cout << "Video time = " << (time_msec() - tstart) << endl;

print_motor_pos(ARM_BOARD);
return_to_start_pos();

reccommand.recording = RECORDING_MARK;
reccommand.marker = 10 * reps + 3;
/*OutPortRecCommandR.Content() = reccommand;
OutPortRecCommandR.Write();
OutPortRecCommandL.Content() = reccommand;
OutPortRecCommandL.Write();*/
sender_RecCommandR.Write(&reccommand);
sender_RecCommandL.Write(&reccommand);

delay(200);

delete_path_array(mpath, dim);
delete_path_array(lpath, dim);
}

cout << "Robot actions finished. Video time = " << (time_msec() - tstart) << endl;

while (time_msec() < (tstart + (VIDEO_FRAMES - 100) * 100L)) {
    delay(500);
}

cout << "Lower head to imitate looking at object. Video time = " << (time_msec() - tstart) << endl;

reccommand.recording = RECORDING_MARK;
reccommand.marker = 7;
/*OutPortRecCommandR.Content() = reccommand;
OutPortRecCommandR.Write();
OutPortRecCommandL.Content() = reccommand;
OutPortRecCommandL.Write();*/
sender_RecCommandR.Write(&reccommand);
sender_RecCommandL.Write(&reccommand);

delay(200);

traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);
m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);

while (time_msec() < (tstart + VIDEO_FRAMES * 100L)) {
    delay(500);
}
cout << "Recording Video has stoped by now. (Video time = " << (time_msec() - tstart) << ")". Wait another 10 sec..." << endl;
cout << "Wait so that node 5 and 6 can save the video!" << endl;
display_wait(60);
display_wait(60);
display_wait(60);
display_wait(60);
cout << "Move head back to upright position." << endl;
traj_deg(HEAD_MOTOR_BASE + NECK_ROLL, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + NECK_YAW, 0, FALSE);
traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 35, FALSE);
traj_deg(HEAD_MOTOR_BASE + EYE_PITCH, 0, FALSE);

m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);

// STOP RECORDING HERE
clipinfo.undistort = 1;
clipinfo.format = PIXEL_BGR;
/*OutPortClipInfoR.Content()=clipinfo;
OutPortClipInfoR.Write();
OutPortClipInfoL.Content()=clipinfo;
OutPortClipInfoL.Write();*/
sender_ClipInfoR.Write(&clipinfo);
sender_ClipInfoL.Write(&clipinfo);
}
else if (!strcmp(buffer, "print"))
{
    print_motor_pos(BOTH_BOARDS);
}
else if (sscanf(buffer, "loaddata %d", &parm1))
{
    ifstream motordata("motordata.dat");
    ifstream objectdata("objectdata.dat");
    int i;
    for(i = 0; i < parm1; i++)
    {
        for(int j=1; j<=ARM_COARSE; j++)
            motordata >> pose[i][j];
        objectdata >> coord[i].x;
        objectdata >> coord[i].y;
        objectdata >> coord[i].z;
        objectdata >> coord[i].xR;
        objectdata >> coord[i].yR;
objectdata >> coord[i].xL;
objectdata >> coord[i].yL;

if(objectdata.eof())
    break;
}

motordata.close();
objectdata.close();

cout << "Loaded " << i << " motor positions." << endl;
} else if (buffer[0]==='d')
{
    int pc = 300;
    int degree = 3;
    int md_type = 0;
    sscanf(buffer, "d %d %d %d", &degree, &pc, &md_type);
    int size;
    int dim;

    long** b_P = load_deg_path_from_file("pathdeg.dat", size, dim, arm_group);
    long** path = new_path_array(pc, dim);

cout << "size=" << size << " dim=" << dim << " " << endl;
    cout << "pcount=" << pc << " " << endl;

cout << "interpolation starts..." << endl;

    if (md_type) {
        spline(size, dim, pc, b_P, path);
    } else {
        bspline(size, dim, pc, degree, b_P, path);
    }

cout << "moving starts..." << endl;

    move_on_path(pc, path, dim, head_group, head_group_addr);

cout << "Press <RETURN> to return to start position." << endl;
    gets(keybuffer);
    print_motor_pos(HEAD_BOARD);
    return_to_start_pos();
    delete_path_array(path, dim);
    delete_path_array(b_P, dim);
}
else if (buffer[0]==='z')
{
    int pc = 500;
    int degree = 3;
    int md_type = 0;
    sscanf(buffer, "z %d %d %d", &degree, &pc, &md_type);

    long** path;

    path = new_path_array(pc + 5, ARM_COARSE);
int b_points = 10;

ifstream msp("path.dat");
msp >> b_points;

for (int i = 0; i < b_points; i++)
    for(int k = 0; k < ARM_COARSE; k++)
        msp >> b_P[k][i];

if (md_type) {
    spline(b_points, ARM_COARSE, pcount, b_P, path);
} else {
    bspline(b_points, ARM_COARSE, pcount, degree, b_P, path);
}

move_on_path(pcount, path, ARM_COARSE, arm_group, arm_group_addr);

cout << endl << "Press <RETURN> to return to start position." << endl;
gets(keybuffer);
print_motor_pos(ARM_BOARD);
return_to_start_pos();

delete_path_array(path, ARM_COARSE);
delete_path_array(b_P, ARM_COARSE);

} else if (buffer[0] == 'y')
{

cout << "Spline path interpolation mode." << endl;

for(int i = 3; i <= ARM_COARSE; i++)
    traj_deg(i + ARM_MOTOR_BASE, 0, FALSE);
m_start_move(arm_group_addr, FALSE);

wait_stop_group(arm_group);
for(int i = 1; i < 2; i++)
    traj_deg(i + ARM_MOTOR_BASE, 0, FALSE);
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

print_motor_pos(ARM_BOARD);
int history_counter = 500;
int md_factor, md_limit, degree;
md_factor = 1;
md_limit = 200;
degree = 3;
int md_type = 0;
sscanf(buffer, "y %d %d %d", &degree, &history_counter, &md_type);

long** path;
path = new_path_array(history_counter + 5, ARM_COARSE);
cout << "history_counter = " << history_counter << endl;

int b_points = 10;
long** b_P;
b_P = new_path_array(b_points, ARM_COARSE);

b_P[0][0] = 0; b_P[1][0] = 0; b_P[2][0] = 0; b_P[3][0] = 0; b_P[4][0] = 0;
b_P[0][1] = 50000; b_P[1][1] = -150000; b_P[2][1] = 0; b_P[3][1] = 50000; b_P[4][1] = 0;
b_P[0][2] = 50000; b_P[1][2] = 0; b_P[2][2] = 0; b_P[3][2] = 50000; b_P[4][2] = 0;
b_P[0][3] = 50000; b_P[1][3] = 0; b_P[2][3] = -40000; b_P[3][3] = 50000; b_P[4][3] = 0;
b_P[0][4] = 50000; b_P[1][4] = 0; b_P[2][4] = -40000; b_P[3][4] = 0; b_P[4][4] = 0;
b_P[0][5] = 0; b_P[1][5] = -150000; b_P[2][5] = 0; b_P[3][5] = 0; b_P[4][5] = 0;
b_P[0][6] = 0; b_P[1][6] = -150000; b_P[2][6] = 0; b_P[3][6] = 10000; b_P[4][6] = 0;
b_P[0][7] = 0; b_P[1][7] = -150000; b_P[2][7] = 0; b_P[3][7] = 0; b_P[4][7] = 0;
b_P[0][8] = 0; b_P[1][8] = -150000; b_P[2][8] = 0; b_P[3][8] = 5000; b_P[4][8] = 0;
b_P[0][9] = 0; b_P[1][9] = 0; b_P[2][9] = 0; b_P[3][9] = 0; b_P[4][9] = 0;

if (md_type) {
    spline(b_points, ARM_COARSE, history_counter, b_P, path);
} else {
    bspline(b_points, ARM_COARSE, history_counter, degree, b_P, path);
}

move_on_path(history_counter, path, ARM_COARSE, arm_group, arm_group_addr);
cout << endl << "Press <RETURN> to return to start position." << endl;
gets(keybuffer);
print_motor_pos(ARM_BOARD);
return_to_start_pos();
delete_path_array(path, ARM_COARSE);
delete_path_array(b_P, ARM_COARSE);
}
else if (sscanf(buffer, "pose %d", &parm1)) {
    pose_index = parm1;
    traj_deg1+ARM_MOTOR_BASE,pose[pose_index][1],FALSE);
    traj_deg2+ARM_MOTOR_BASE,pose[pose_index][2],FALSE);
    traj_deg3+ARM_MOTOR_BASE,pose[pose_index][3],FALSE);
    traj_deg4+ARM_MOTOR_BASE,pose[pose_index][4],FALSE);
    traj_deg5+ARM_MOTOR_BASE,pose[pose_index][5],FALSE);
m_start_move(arm_group_addr, FALSE);
wait_stop_group(arm_group);

cout << "Rec object info: " << coord[pose_index].xR << ", " << coord[pose_index].yR << ", "
     << coord[pose_index].xL << ", " << coord[pose_index].yL << ", "
     << coord[pose_index].x << ", " << coord[pose_index].y << ", " << coord[pose_index].z << endl;

Coord3D coord_now;
delay(1000);
read_depth(0, &coord_now, NULL);

cout << "Real object info: " << coord_now.xR << ", " << coord_now.yR << ", "
     << coord_now.xL << ", " << coord_now.yL << ", "
     << coord_now.x << ", " << coord_now.y << ", " << coord_now.z << endl;

} else if (!strcmp(buffer, "quit"))
{
    cout << "Quit application." << endl;
    break;
}
else if (!strcmp(buffer, "kinemode"))
{
    cout << "Starting kine learning mode." << endl;
    kine_learning();
    cout << "Back to weird-mode." << endl;
}
else if (!strcmp(buffer, "fastkinemode"))
{
    cout << "Starting fast kine learning mode." << endl;
    kine_learning_fast_without_user_input();
    cout << "Back to weird-mode." << endl;
}
else
{
    cout << "Unknown command. Type help for a summary of valid commands!" << endl;
}

delay(200);
num_trials++;

} return_to_start_pos();
print_motor_pos(ARM_BOARD);
return 0;

static void terminate(int sig)
{
    cout << "Terminate: ";
    // wait_stop_group(arm_group);
    return_to_start_pos();
cout << "ARM ";
//
wait_stop_group(head_group);
return_head_to_start_pos();
cout << "HEAD." << endl;

m_destroy();

sender_RecCommandL.Stop();
sender_RecCommandR.Stop();
sender_ClipInfoL.Stop();
sender_ClipInfoR.Stop();
receiver_Depth_Info.DisconnectAll();

cout << "All ports closed. EXIT(0)." << endl;
exit(0);
}

int main(int argc, char *argv[])
{
    char portname[100];

cout << "Headarm reporting for duty!" << endl;
cout.flush();

    int receiver_Depth_Info_ip = 1;
    int receiver_Depth_Info_port = 5000;
    int receiver_Depth_Info_enable = 0;
    int sender_ClipInfoL_port = 5001;
    int sender_ClipInfoR_port = 5002;
    int sender_RecCommandL_port = 5003;
    int sender_RecCommandR_port = 5004;
    int motorlib_ip0 = 1;
    int motorlib_port0 = 5200;
    int motorlib_ip1 = 1;
    int motorlib_port1 = 5300;

    char c;
    while((c=getopt(argc, argv, "D:C:M:n:m:a?")) != -1)
    {
        switch(c)
        {
        case 'n': Num_Random_Pose = atoi(optarg);
            if(Num_Random_Pose>RANDOM_POSE_MAX || Num_Random_Pose<0)
                Num_Random_Pose=RANDOM_POSE_MAX;
            break;

        case 'm': Distance_Mode = atoi(optarg);
            if(Distance_Mode!=WINDOW_MODE && Distance_Mode!=THREED_MODE)
                Distance_Mode=WINDOW_MODE;
            break;

        case 'a': Averaging_Neighbors = true;
            break;

        case 'C': sscanf(optarg, "%d,%d", &sender_ClipInfoL_port,
&sender_ClipInfoR_port);
            break;

        case 'R': sscanf(optarg, "%d,%d", &sender_RecCommandL_port,
&sender_RecCommandR_port);
            break;

        default: break;
        }
    }

    return 0;
}
case 'D':
    sscanf(optarg, "%d:%d", &receiver_Depth_Info_ip, &receiver_Depth_Info_port);
    receiver_Depth_Info_enable = 1;
    break;

case 'M':
    sscanf(optarg, "%d:%d,%d:%d", &motorlib_ip0, &motorlib_port0, &motorlib_ip1, &motorlib_port1);
    break;

case '?':
    cout << "Usage: headarm -n -m -a -?" << endl;
    exit(0);
    break;

default: break;
}

sender_RecCommandL.Start(sender_RecCommandL_port);
sender_RecCommandR.Start(sender_RecCommandR_port);
sender_ClipInfoL.Start(sender_ClipInfoL_port);
sender_ClipInfoR.Start(sender_ClipInfoR_port);

char str_tmp[20];
if (receiver_Depth_Info_enable)
{
    sprintf(str_tmp, "192.168.0.%d", receiver_Depth_Info_ip);
    receiver_Depth_Info.ConnectTo(str_tmp, receiver_Depth_Info_port);
}

cout << "Setting up motor ports..."; 
char str_tmp0[20]; 
char str_tmp1[20]; 
char *motor_addresses[2]; 
sprintf(str_tmp0, "192.168.0.%d", motorlib_ip0); 
sprintf(str_tmp1, "192.168.0.%d", motorlib_ip1); 
motor_addresses[0] = str_tmp0; 
int motor_iports[2]; 
motor_iports[0] = motorlib_port0; 
motor_iports[1] = motorlib_port1; 
m_init(motor_addresses, motor_iports, NUM_MOTORS, true, true, true, true, true);

cout << "success." << endl;

cout << "Performing test read of arm position..." << endl;
print_motor_pos(ARM_BOARD);
cout << "Position reading successful" << endl;

cout << "Performing test read of head position..." << endl;
print_motor_pos(HEAD_BOARD);
cout << "Position reading successful" << endl;

//All motors must be stopped first using AMP_EN before any command can be issued
cout << "Stopping motors..."; 
stopAllMotors(ARM_BOARD); 
stopAllMotors(HEAD_BOARD); 
cout << "successful" << endl;

//Reset motor positions
for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
    m_reset_pos(i, false);
for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
    m_reset_pos(i, false);

int arm_group_index=0;
int head_group_index=0;
int is_leader;

for(int i=ARM_MOTOR_BASE+1; i<=ARM_MOTOR_BASE+ARM_MOTORS; i++)
{
    arm_group[arm_group_index++]=i;
    if(i==ARM_MOTOR_BASE+1)
        is_leader=1;
    else
        is_leader=0;
    m_set_group_addr(i, arm_group_addr, is_leader);
}
arm_group[arm_group_index]=GROUP_END;

arm_group_index=0;
while(arm_group[arm_group_index]!=GROUP_END)
    cout << arm_group[arm_group_index++] << " ";
    cout << endl;

for(int i=HEAD_MOTOR_BASE+1; i<=HEAD_MOTOR_BASE+HEAD_MOTORS; i++)
{
    head_group[head_group_index++]=i;
    if(i==HEAD_MOTOR_BASE+1)
        is_leader=1;
    else
        is_leader=0;
    m_set_group_addr(i, head_group_addr, is_leader);
}
head_group[head_group_index]=GROUP_END;

head_group_index=0;
while(head_group[head_group_index]!=GROUP_END)
    cout << head_group[head_group_index++] << " ";
    cout << endl;

signal(SIGTERM, &terminate);
signal(SIGINT, &terminate);

// kine_learning();

traj_deg(HEAD_MOTOR_BASE + UPPERNECK_PITCH, 35, FALSE);
m_start_move(head_group_addr, FALSE);
wait_stop_group(head_group);
m_reset_pos(HEAD_MOTOR_BASE + UPPERNECK_PITCH, false);

make_moves();

return 0;
Tasks.cc

Coded by Erica Baller
4/24/06

Tasks.cc in ebb23@cs.zoo.yale.edu:/FinalP490/Tasks.cc
Includes all code for tasks, values randomly generated by computer or
coder to later be linked to porters for motors in arm and visual
perception (through new grabber)*

#include <iostream.h>
#include <fstream.h>
#include <algorithm>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <math.h>
/*#include "libmotor.h"
#include "attn.h"
#include "newdepth.h"
#include "motion.h"
#include "cam_data1.h"
#include "cam_data2.h"
#include "newgrabber.h"
*/

#include "libmotor.h"
#include "attn.h"
#include "newdepth.h"
#include "motion.h"
#include "cam_data1.h"
#include "cam_data2.h"
#include "newgrabber.h"
*
/*#include "PortOf.h"
#include "cam_data1.h"
#include "cam_data2.h"
#include "newgrabber.h"
#include "Framegrabber.h"
#include "ImageSendable.h"*/

using std::string;

//--------------------------------------------------------------
// Process name
char *my_name = "/secondstephead";
//--------------------------------------------------------------
//The ratio of motor servo counts to seconds, # of encoder ticks per revolution
#define SPEEDRATIO 1953
#define ENCODERRATIO 2048

#define NUM_MOTORS 6
//--------------------------------------------------------------
//debugging variable
#define debug 1
#define tableHeight 0.5 //meters?
#define startPosx 0.0 // in meters?
#define startPosy 1.0 // in meters?
#define startPosz 0.5// in meters?

//--------------------------------------------------------------
//--------------------------------------------------------------
//Helper Functions------------------------------------------------
int MaxMin (double & num1, double & num2)
/*takes num1 and num2, swaps if num1 is greater than num2 and returns 1 on
swap, 0 on no swap. Max will always be num1 at end of function*/
{
    if (num1 < num2) //swap
    {
        double temp = num1;
        num1 = num2;
        num2 = temp;
        return 1;
    }
    //leave alone otherwise
    return 0;
}

double Max (double i, double j)
{
    return ((i > j) ? i : j);
}

double Min (double i, double j)
{
    return ((i < j) ? i : j);
}

void Swap (double & i, double & j)
{
    double temp = i;
    i = j;
    j = temp;
}

int TapeLocations (int & redSalience1, int & redSalience2,
    double & y1, double & y2, double & x1, double & x2)
{
    //Send pic through color, analyze red saliency of boxes
    //for now
    redSalience1 = rand() % 2; //sub with sending picture through filter and
    //and the salience value it returns
    redSalience2 = rand() % 2;

    //if ((redSalience1 && redSalience2) != 0 ) they are salient enough, later
    //a threshold value to be > will sub
    if (1)
    {
        //to be replaced by the upper bound on the salience boxes
        y1 = rand() % 100 + 15.0; //upperbound on 1st red salient object
        y2 = rand() % 100 + 15.0; //upperbound on 2nd red salient object
        x1 = rand() % 100 + 25.0; //rightbound on 1st red salient object
        x2 = rand() % 100 + 25.0; //rightbound on 2nd red salient object
    }
if (debug == 1)
{
    cout << "before MaxMin, possible swap\n";
    printf("x1 = %e, x2 = %e, y1 = %e, y2 = %e\n",
            x1, x2, y1, y2);
}

if (MaxMin(y2, y1) == 1) //swap occurred since y2 is greater than y1
{
    //keep groups together
    if (debug == 1)
    {
        cout << "In MaxMin, x's should swap  Preswap\n";
        printf ("x1 = %e, x2 = %e, y1 = %e y2 = %e\n", x1, x2, y1, y2);
    }
    Swap (x1, x2);
    if (debug == 1)
    {
        cout << "In MaxMin, x's should have swapped  Postswap\n";
        printf ("x1 = %e, x2 = %e, y1 = %e y2 = %e\n", x1, x2, y1, y2);
    }
}
else
{
    y1 = y2 = x1 = x2 = -1.0;
    return 0;
}

// later will be sending the grabber picture too and color representation-->
//  2 - red
//  3 - green
//  5 - yellow (bright)
// currently randomly generates coordinates for the "color " blob
int ColorBlob(double & y, double & x, int color)
{
    int found = 1;
    if (debug == 0)
        found = ((rand() % color) == 0) ? 1 : 0; //must be replaced
    //randomly tells you the color blob is "found" when mod returns no remainder
    if (found == 1)
    {
        //blob was "found"
        y = rand() % 100 * 1.0;
        x = rand() % 100 * 1.0;
    }
    else
        y = x = -1.0;

    return found;
}

int LineSegment (double & m, double & b, double & y1, double & y2, double & x1, double & x2)
//image will be sent)
//creates a line segment from salient red blobs in T(1)
//m is slope, b is intercept
{
    int redSalience1, redSalience2;
if (TapeLocations(redSalience1, redSalience2, y1, y2, x1, x2) != 0) //draw line, rsR is x-axis  
    // rsU is y-axis
    {
        m = (y2 - y1)/(x2 - x1);
        b = y2 - (m * x2);
        return 1;
    }
else
    {
        //no line segment could be drawn
        m = -1;
        b = -1;
        return 0;
    }
}

int LocationProg(double prevx, double prevy, double & currx, double & curry, double & yProg, double & xProg, int color)
//identifies if person has made positive or negative progression to complete
//task for T1 right now
{
    if (ColorBlob(curry, currx, color) == 1)
    {
        yProg = prevy - curry; //if +, should be + or zero with a bit of flex.
        xProg = currx - prevx; //ditto, except currx should be larger
        return 1;
    }
else
    return 0; //color not found
}

int TaskCompletion1()
{
    double prevx, prevy, currx, curry, yProg, xProg; //vars for person
    double y1, y2, x1, x2; //vars for line
    double m, b;
    int y, x, color, colorP; //for any specific coordinates of a specific color
    int loopCount = 0; //how many iterations of behavior/reinforcement
    int done = 0; //loop control, sets to done if completion or loopCount > 20
    double distanceAboveTable; //initial y coords above table, can't get higher
    double midx, midy, temp;
    int typeReinf, numBehaviors, intensity; //type is 0 for nothing -1 for shake
    //1 for Nod
    // NumBehaviors is how many shake/nods
    // intensity is how powerful

cout << "debug = " << debug << endl;
if (debug == 1)
{
    cout << "initializing vars in TaskCompletion1\n";
}
//initialize
if (LineSegment(m, b, y1, y2, x1, x2))
{
    if (debug)
    {
        cout << "In line segment, completed successfully\n";
        cout << "Resulting Values ==>\n";
        printf("m = %e, b = %e, x1 = %e, x2 = %e, y1 = %e, y2 = %e\n", m, b, x1, x2, y1, y2);
    }
prevx = prevy = 0;
yProg = xProg = 0;
typeReinf = numBehaviors = intensity = 0;
//Will be removed and replaced with real things--------
// y1 = rand() % 100 + 15; upperbound on 1st red salient object
//y2 = rand() % 100 + 15; upperbound on 2nd red salient object
//x1 = rand() % 100 + 25; rightbound on 1st red salient object
//x2 = rand() % 100 + 25; rightbound on 2nd red salient object
midx = (fabs(x2-x1)/2);
my = (m * midx) + b;
temp = (rand() % 100) * 1.0;
currx = (midx > (temp + 25)) ? temp : 1.0;
temp = (rand() % 100) * 1.0;
curry = (my < (temp + 25)) ? temp : (my + 30);
distanceAboveTable = curry;

switch (rand() % 3)
{
    case 0: colorP = 2;
        break;
    case 1: colorP = 3;
        break;
    default : colorP = 5;
}

if (debug == 1)
{
    cout << "All initialized: done = " << done << endl;
    printf("prevx = %e, prevy = %e, currx = %e, curry = %e, xProg = %e, yProg = %e\n", prevx, prevy, currx, curry, xProg, yProg);
    printf("colorP = %d, typeReinf = %d, numBehaviors = %d, intensity = %d\n",colorP, typeReinf, numBehaviors, intensity);
    printf("x = %d, y = %d, x1 = %d, x2 = %d, y1 = %d, y2 = %d\n", x, y, x1, x2, y1, y2);
    printf("m = %e, b = %e, midx = %e, midy = %e, temp = %e, distanceAboveTable = %e\n", m, b, midx, my, temp, distanceAboveTable);
}

while ((loopCount < 20) && (done == 0))
{
    loopCount++;
    if (debug == 1)
    {
        cout << "loopcount = " << loopCount << endl;
        printf("prevx = %e, prevy = %e\n", prevx, prevy);
    }

    if (LocationProg(prevx, prevy, currx, curry, yProg, xProg, colorP) == 1)
    {
        if (debug == 1)
        {
            cout <<"LocationProg was successful in finding the right Color ";
            cout << colorP << endl;
        }

        if (yProg >= 0)
        {
            if (debug == 1)
if (xProg >= 0) {
    double maxX = midx;
    if (debug == 1)
        printf("xProg is positive or 0, xProg = %e, xProg\n", xProg);
}
if (currx > (maxX + 5))//arbitrary, passed the table
    //Shake(5, currx - (maxX-midx)) fast quick shakes
    typeReinf = -1;
    numBehaviors = int(currx - (maxX - midx));
    intensity = int(pow(numBehaviors, 2));
    if (debug == 1)
        {cout << "currx has gone PAST the table, no good\n";
         printf("currx = %e, maxX = %e, reinf = %d, numBehaviors = %d, intensity = %d\n", currx,
         maxX, typeReinf, numBehaviors, intensity);)
}
else
    if (curry < midy) // you are low enough
        if (debug == 1)
            printf("curry is less than midy, midy = %e, curry = %e\n", midy, curry);
        if (int(fabs(midx - currx)) < 10)//you are close enough
            //and under table and DONE
            //clap?
            typeReinf = 0;
            numBehaviors = 0;
            intensity = 0;
            done = 1;
            if (debug == 1)
                {cout << "You GOT IT!!!\n";
                }
else
            if (debug == 1)
                {cout << "Squat but not close enough \n";
                }
        else
            if (debug == 1)
                {cout << "Nod((xProg + yProg)/3, or some num to divide by
                // sqrt(pow(yProg, 2) + pow(xProg, 2)))
                typeReinf = ((yProg + xProg) > 0) ? 1 : -1;
                numBehaviors = int(fabs((xProg + yProg)/3));
                intensity = int(sqrt(int(yProg * yProg) + int(xProg * xProg)));
                }
else {
    temp = fabs(xProg - yProg);
    if (yProg < 0) //HUGE mistake was made
        //Nod((xProg + yProg)/3, or some num to divide by
        //sqrt(pow(yProg, 2) + pow(xProg, 2));
        typeReinf = -1;
        numBehaviors = int((xProg + yProg)/3;
        intensity = int(sqrt(pow(yProg, 2) + pow(xProg, 2))); if (debug) {
            cout << "Bad Run yProg == " << yProg;
            cout << "xProg == " << xProg << endl;
        }
    else if (temp > 3) {
        //Shake(temp, pow(temp, 2));
        typeReinf = -1;
        numBehaviors = int(temp);
        intensity = int(pow(numBehaviors, 2));
        if (debug) {
            cout << "Only a bit bad yProg == " << yProg;
            cout << "xProg == " << xProg << endl;
        }
    } else if (curry > (distanceAboveTable)) {
        //Shake(5, curry - (fabs(midy-y2)) fast quick shakes
        typeReinf = -1;
        numBehaviors = int(curry - distanceAboveTable);
        intensity = int(pow(numBehaviors, 2));
        if (debug) {
            cout << "curry is too BIG curry == " << curry << endl;
        }
    }
}
if (done) {
    if (debug) {
        cout << "Still task to go ...
";
    }
    if (typeReinf == -1) {
        if (debug) {
            cout << "Negative Feedback\n";
        }
        //Shake(numBehaviors, intensity);
    } else {
        if (debug) {
            
        }
    }
cout << "Positive Feedbackn";

//Nod(numBehaviors, intensity);

else
{
    if (debug)
        cout << "Task Compete"
    //Do behaviors for completion
    loopCount == 100;
}

if (debug)
    printf("Concluding feedback = %d, numBehaviors = %d, intensity = %d
", typeReinf, numBehaviors,
    intensity);

prevy = curry;
prevx = currx;

if (debug)
    cout << "loopcount ==> " << loopCount << endl;

return (loopCount < 20);

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
//­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­
/*void setupMotorPos()
{
    for(int i=1; i<=NUM_MOTORS; i++)
        m_reset_pos(i, false);
}*/
//­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­
/*void stopAllMotors ()
{
    for(int i=1; i<=NUM_MOTORS; i++)
        m_stop_smooth (i);
}

/*void print_motor_pos()
{
    long pos[NUM_MOTORS+1];
    for(int i=1; i<=NUM_MOTORS; i++)
        m_get_pos(i,pos[i],true);
    delay(500);
    cout<<pos[i];
    if(i!=NUM_MOTORS)
        cout<<", ";;
    else
        cout<<endl;
}
}*/
//­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­
Positive Feedback

/* Given when someone progresses to a new stage, completes a stage or completes 
the given task */

//­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­
/*void Nod(int reps, int intensity) 
{
    int done = 0;
    if (debug)
        cout << "In and beginning Nod ...
" << endl;
    for (int i = 0; i < reps; i++)
    {
        m_traj_position(3, 40000, 20.0 * intensity, 0.02 * intensity, TRUE);
        delay(550/intensity);
        m_traj_position(3, 0, 20.0 * intensity, 0.02 * intensity, TRUE);
        delay(550/intensity);
    }
    if (debug)
        cout << "Exiting Nod ...
" << endl;
    return done;
}*/

Negative Feedback

/* Exerted when a person has not completed or reached a particular stage, 
when they go back a stage, or do something entirely inappropriate */

//­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­
/*void Shake(int reps, int intensity) 
{
    int done = 0;
    if (debug)
        cout << "In and beginning Shake ...
" << endl;
    for (int i = 0; i < reps; i++)
    {
        m_traj_position(2, 50000, 20.0 * intensity, 0.02 * intensity, TRUE);
        delay(1000/intensity);
        m_traj_position(2, 0, 20.0 * intensity, 0.02 * intensity, TRUE);
        delay(1000/intensity);
        m_traj_position(2, -50000, 20.0 * intensity, 0.02 * intensity, TRUE);
        delay(1000/intensity);
    }
    if (debug)
        cout << "Exiting Shake ...
" << endl;
    return done;
}*/
int ArmToPerson()
    /* will point at person, straight ahead, may take perceptual info */
    { int done = 0;
        return done;
    }

int ArmToTable()
    /* will point under table, taking perceptual cues */
    { int done = 0;
        return done;
    }

int GetUnderTable()
    /* only performed if person at table, tells them to get under table
        arm up, wrist bent forward—arm down, wrist arches — 1 movement */
    { int done = 0;
        return done;
    }

int CrouchedPos()
    /* takes feedback from grabber, performed if person is crouched but not
        under the table. 2 possibilities: 1) head turns to table, chin down then up
        twice or 2) wrist flick, away from table, toward table, x2 */
    { int done = 0;
        return done;
    }

int Distance (float midxt, float midyt, float & x, float & y, float & z)
    // white blob
    { z = startPosz;
        return sqrt((midxt *}
    }

int Task1()
    { // get line and distance to table
        float rt1x, rt1y, rt2x, rt2y, mt, bt;
        if (LineSegment(mt, bt, rt1y, rt2y, rt1x, rt2x) == 1)
        { float midxt, midyt;
            midxt = (rtx1 + rtx2)/2;
            midyt = (rty1 + rty2)/2;
            float distanceToTable = Distance(midxt, midyt);
            }
    }
int InitialNicoMovement1()
{
    int done = 0; //returns 1 if nico does movement, 0 if he does not or fails
    int rNum = rand() % 2; //for generating whether or not move was a success
    if (debug == 1)
    {
        if (rNum == 0)
        {
            cout << "Nico could not perform InitialMoveOne successfully";
            cout << " and can not run the test.\n\nSorry\n";
        }
        else
        {
            cout << "Nico performed InitialMoveOne successfully";
            cout << " and will now continue the test.\n"
        }
    }
    done = rNum;
    return done;
}
//-----------------------------------------------

int InitialNicoMovement2()
{
    int done = 0; //returns 1 if nico does movement, 0 if he does not or fails
    int rNum = rand() % 2; //for generating whether or not move was a success
    if (debug == 1)
    {
        if (rNum == 0)
        {
            cout << "Nico could not perform InitialMoveTwo successfully";
            cout << " and can not run the test.\n\nSorry\n";
        }
        else
        {
            cout << "Nico performed InitialMoveTwo successfully";
            cout << " and will now continue the test.\n"
        }
    }
    return done;
}
done = rNum;
return done;
} //-------------------------------------------------------------------------------------------
int InitialNicoMovement3()
{
    int done = 0; //returns 1 if nico does movement, 0 if he does not or fails
    int rNum = rand() % 2; /*for generating whether or not move was a success
    before connecting to porter*/
    if (debug == 1)
    {
        if (rNum == 0)
        {
            cout << "Nico could not perform InitialMoveThree successfully";
            cout << " and can not run the test.\nSorry!\n";
        }
        else
        {
            cout << "Nico performed InitialMoveThree successfully";
            cout << " and will now continue the test.\n";
        }
    }
    done = rNum;
    return done;
} //-------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------
int TaskOne() /* This task requires that a person get under a table*/
{
    int success = 0; //if they succeed at task, turns into a 1
    success = InitialNicoMovement1();
    if (success == 1)
    {
        TaskCompletion1();
    }
    return success;
} //-------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------
int TaskTwo() //never used or implemented
/* This task requires that a person picks up a ball and bounces it*/
{
    int success = 0; //if they succeed at task, turns into a 1
    success = InitialNicoMovement2();
    return success;
} //-------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------
int TaskThree() //never used or implemented
/* This task requires that a person put a red ball inside a green box*/
{
    int success = 0; //if they succeed at task, turns into a 1
    success = InitialNicoMovement3();
    return success;
} //-------------------------------------------------------------------------------------------
int main ()
{
srand(time(NULL));
string s;
char ch;
int success;
Task1();
do {
    int randNum = rand() % 3;
    switch (randNum) {
        case 0: success = TaskOne();
            break;
        case 1: success = TaskTwo();
            break;
        default: success = TaskThree();
    }
    if (debug == 1) {
        cout << "Success value from Task ";
        cout << randNum + 1 << " ==> " << success << endl;
    }
    if (success == 0) {
        if (debug == 1) {
            cout << "Do you still want to continue ";
            cout << "though Nico sucks and success is 0? (y/n) ==> ";
            getline(cin, s);
            ch = tolower(s[0]); //To ensure the character is case-specific
        } else
            ch = 'n';
    } else {
        cout << "Do you want to continue (Nico success is 1)? (y/n) ==> ";
        getline(cin, s);
        ch = tolower(s[0]); //To ensure the character is case-specific
    }
    while (ch == 'y');
}

Thank you for playing
return 0;

///////////////////////////////////////////////////////////////////////////////////

Depth.cc

/*Depth.cc in ebb23@cs.zoo.yale.edu:/FinalP490/Depth.cc
Written by Erica Baller April 2005
This code takes manually defined constants representing two items tagged by color.h and color.cc in the form of pixel coordinates from images generated by both the right and the left eye in cam. It calculates the depth of each object and the line connecting both objects. Used to calculate the coordinates and height of the table (implemented to ease such calculations in case the table is moved and its position must quickly be recalculated)
*/
#include <iostream.h>
#include <fstream.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sched.h>
#include <time.h>
#include <ctype.h>
#include <fcntl.h>
#include <math.h>

// Constants

#define PB_WIDTH 304
#define PB_HEIGHT 224
#define PB_MIDX 152
#define PB_MIDY 112
#define RNX 216
#define RFX 260
#define RNY 180
#define RFY 189
#define LNX 219
#define LFX 267
#define LNY 182
#define LFY 191

struct Blob
{
  float x;
  float y;
  float m_through_mid;
  float b_through_mid;
  float dist_from_xmid;
  float hypot;
};

typedef struct Blob Blob;

struct DepthBlob
{
  Blob r;
  Blob l;
  float x;
  float y;
  float z;
};

typedef struct DepthBlob DepthBlob;

struct TwoBlobs
{
  DepthBlob image_n;
  DepthBlob image_f;
  float m;
  float b;
  float y; //height off the ground
};

typedef struct TwoBlobs TwoBlobs;

void CalcTwoBlobs (TwoBlobs *tb)
{ 
    tb -> m = ((tb -> image_f.y - tb -> image_n.y)/
             (tb -> image_f.x - tb -> image_n.x));
    tb -> b = tb -> image_f.y - (tb -> m * tb -> image_f.x); 
    
    tb -> y = tb -> image_f.y;
}

//----------------------------------------------------------------------
void CalcDepthBlob(DepthBlob *db)
{
    db -> x = (float) ((db -> r.b_through_mid - db -> l.b_through_mid)/
                       (db -> l.m_through_mid - db -> r.m_through_mid));
    db -> y = (db -> r.m_through_mid * db -> x) + db -> r.b_through_mid;
    db -> z = db -> y - db -> r.y;
}

//----------------------------------------------------------------------
void CalcBlob(Blob *aBlob)
{
    aBlob -> m_through_mid = (aBlob -> y - PB_MIDY)/(aBlob -> x - PB_MIDX);
    aBlob -> b_through_mid = aBlob -> y - (aBlob -> m_through_mid * aBlob -> x);
    aBlob -> dist_from_xmid = PB_MIDX - aBlob -> x;
    aBlob -> hypot = sqrt(pow((aBlob -> x - PB_MIDX), 2) +
                          pow((aBlob -> y - PB_MIDY), 2));
}

//----------------------------------------------------------------------
int main (int argc, char *argv[])
{
    TwoBlobs AllBlobsInfo;
    AllBlobsInfo.image_n.r.x = RNX; 
    AllBlobsInfo.image_f.r.x = RFX;
    AllBlobsInfo.image_n.r.y = RNY; 
    AllBlobsInfo.image_f.r.y = RFY;
    AllBlobsInfo.image_n.l.x = LNX;
    AllBlobsInfo.image_f.l.x = LFX;
    AllBlobsInfo.image_n.l.y = LNY;
    AllBlobsInfo.image_f.l.y = LFY;
}
CalcBlob(&AllBlobsInfo.image_n.r);
CalcBlob(&AllBlobsInfo.image_n.l);
CalcBlob(&AllBlobsInfo.image_f.r);
CalcBlob(&AllBlobsInfo.image_f.l);
CalcDepthBlob(&AllBlobsInfo.image_n);
CalcDepthBlob(&AllBlobsInfo.image_f);
CalcTwoBlobs(&AllBlobsInfo);

cout << "will go nr x fr x nl x fr y nly fry\n";
cout << AllBlobsInfo.image_n.r.x << " " << AllBlobsInfo.image_f.r.x << endl;
cout << AllBlobsInfo.image_n.l.x << " " << AllBlobsInfo.image_f.l.x << endl;
cout << AllBlobsInfo.image_n.r.y << " " << AllBlobsInfo.image_f.r.y << endl;
cout << AllBlobsInfo.image_n.l.y << " " << AllBlobsInfo.image_f.l.y << endl;
}

//------------------------------------------------