C++ Software for the Kinereach

to be used with

Microsoft Visual Studio 2010 Express,

Windows 7,

and Matlab 2012b (or higher)

Instructions for

General Setup and Execution

C++ software written by:

Adrian Haith

Promit Roy

Aneesh Roy

Aaron Wong

System Calibration Routine by:

Aaron Wong

1. Getting Started

1.2 Running an existing project

1.2 Creating a new project in MS Visual Studio

1.3 General organization structure of the code

1.4 Data recording

1.5 Connecting the Kinereach for the first time

1.6 Calibrating the Kinereach

2. Programming manual

2.1 Configuration

2.1.1 Constants

2.1.2 COM ports

2.2 Application (main.cpp)

2.2.1 Methods

2.2.2 Loading files

2.2.3 Displaying objects

2.2.4 Playing sounds

2.2.5 Reading inputs

2.3 Classes

2.3.1 InputFrame

2.3.2 TrackBird (FOB and TrakStar compatible)

2.3.3 MouseInput

2.3.5 JoystickInput

2.3.6 DataWriter

2.3.7 Sound

2.3.8 Image

2.3.9 Object2D

2.3.10 Region2D

2.3.10 Path2D

2.3.11 Circle

2.3.12 HandCursor

2.3.13 Ftdi

**1. Getting Started**

**1.1 Running an existing project**

In general, this code is run in “debug” mode, as this method is safest for catching any potential errors that crop up both during programming and also during testing (e.g., if a parameter is incorrectly set or a trial table does not load). As the project runs, notes will be occasionally logged in the standard error output (e.g, stderr.txt in the Debug folder). These notes reflect only the most current execution of the program, so if there is any desire to save this information the text file should be copied into another directory.

The typical method of running an existing project is as follows.

A. The project should be opened in visual studio.

B. Any free parameters are set in config.h, and the trial table for the current block must be specified in the trialtable.txt file that exists in the project directory (see section 1.3 for details). Be sure to save trialtable.txt after making any modifications, or the new trial table will not be loaded.

C. If hand tracking will be performed with the Ascension tracking system (Flock of Birds or the TrakStar), the birds should be connected and turned on; otherwise, the program will default to mouse emulation mode.

D. The display should be connected to the computer, and turned on.

E. Hit the run button on the menu bar. The paradigm should execute as intended. To quit early before the conclusion of the trial table, hit the “escape” key.

F. Data should be automatically recorded into a unique file that is time stamped, and will appear in the specified data directory (set in config.h).

**1.2 Creating a new project in MS Visual Studio**

This code has been written in Microsoft Visual Studio 2010 Express; as such, it should be opened and executed only in Visual Studio 2010.

Open MS Visual Studio. Select “New project.”

Choose to create a Win32 Project.

Then, select the path into which you want a new folder created, and name it as desired.

* Note that this will automatically generate a folder with the selected name in the selected directory. Make sure that the "Create directory for solution" option is unchecked (this will merely add an additional folder in the project directory that will contain only the project solution file and separate it from the project file, which just gets confusing).
* Reference: The $(ProjectDir) is going to be the $(path)\$(project name).

Choose OK. Click Next.

Under “Application type”, chose “Windows application.”

Under “Additional Options”, choose “Empty project.”

Click Finish.

In Windows explorer, Navigate to the newly created project folder on the hard drive, and place a copy all \*.cpp files, \*.h files, and resource folders from a previous project into that folder.

Also while in the Windows explorer, ensure that a copy of all the .dll files exist in the Debug folder. This can generally be achieved by ensuring that a copy of the Debug folder from the previously existing project has been copied into the current project directory; otherwise, it requires coping the .dll files out of the /Flock\_of\_Birds, /TrakStar, and /SDL-1.2.15/lib/x86 folders.

In the new windows project, look at the solution explorer window. Right click on “Source Files”, and choose add existing items. Add all the .cpp files that will be used in this current project (at minimum, this requires main.cpp).

Similarly, right click “Header Files” and choose add existing items, selecting the desired .h files.

In the solution explorer window, right click on the project name and choose “properties”. This opens the properties page. Set the following options:

1) Configuration Properties -> VC++ Directories -> Include Directories, add the following

$(ProjectDir)\SDL-1.2.15\include

$(ProjectDir)\Flock\_of\_Birds

$(ProjectDir)\TrakStar

2) Configuration Properties -> VC++ Directories -> Library Directories, add the following:

$(ProjectDir)\SDL-1.2.15\lib\x86

$(ProjectDir)\Flock\_of\_Birds

$(ProjectDir)\TrakStar

3) Configuration Properties -> C/C++ -> Additional Include Directories, add the following

$(ProjectDir)\SDL-1.2.15\include

$(ProjectDir)\Flock\_of\_Birds

$(ProjectDir)\TrakStar

2) Configuration Properties -> Linker -> Additional Library Directories, add the following:

$(ProjectDir)\SDL-1.2.15\lib\x86

$(ProjectDir)\Flock\_of\_Birds

$(ProjectDir)\TrakStar

4) Configuration Properties -> Linker -> Input -> Additional Dependencies, add the following:

opengl32.lib

glu32.lib

SDL.lib

SDLmain.lib

SDL\_mixer.lib

SDL\_ttf.lib

SDL\_image.lib

Bird.lib

ATC3DG.lib

Click Apply, and OK to close the properties page.

Save this project. In the future, this project can be retrieved by choosing to open the existing Project (note, you have to open the project, not the solution).

**1.3 General organization structure of the code**

The main body of the code is contained in main.cpp. This code is managed by a loop in main() that has 4 jobs: (1) look for inputs, either from the keyboard or from the hand (via the Ascension or a mouse), (2) write out data to the data file, (3) execute the current state (more on this below), and (4) update the display.

At the heart of this code is a state machine, which checks the status of various flags and parameters (and the availability of new data samples) on each pass through the code, and sets appropriate changes in state or updates various parameters in response. In this way, the code does not halt and wait for any particular event, which eliminates the danger of losing a large number of data samples while the code waits for the event to occur. The current state of the state machine is set locally by each state. All statements in the current state are executed on each iteration of the main loop until the next state is selected; any operations that should be performed only upon entry into or exit from the state should therefore be housed within a conditional (if) statement.

Main.cpp also contains additional small functions that control things like reading the trial table (see below) or controlling what gets displayed to the screen. There are three additional but critical functions: an initialization function that is run immediately upon startup, a function to initialize the Ascension, and a function to neatly shut down everything at the end of the block (which frees up any allocated memory appropriately).

This code runs according to a trial table file, which contains the set of parameters that are necessary to describe what occurs on a given trial. Each line in the text-based trial table file contains information for one trial. This file is read in once during initialization, and is stored in an array which can be both read and modified throughout the experiment. Generally, the trial table houses information about the locations and timing of targets to be displayed during the trial. The name of the trial table is specified as a parameter in config.h. It is important that the extension also be included in the name of the trial table file (e.g., .txt).

The majority of control parameters not specified in the trial table are provided in config.h as defined constants. In general, these constants should not need to be changed frequently.

This code also relies on a number of objects (e.g., associated .h and .cpp files) that are useful for performing various specific functions. Many of these are related to the display of particular things, such as pictures and text (Image, Object2D), circles (Circle), hand cursors (HandCurs) polygonal regions (Region2D), line paths (Path2D), and sounds (Sound). Others are related to data input (TrackBird, InputFrame, MouseInput, TargetFrame) or output (DataWriter). Additional objects may be included or written as desired.

**1.4 Data recording**

Depending on the particular version of code being used, data may be stored in one of two ways. In general, data are recorded in a folder that is specified in config.h, and are parsed such that each file contains data from one reach, defined as movement from the start target outward and then back to the start position. Each data file is identified with both a time stamp and a name indicating the trial number whose data is recorded in that file. Once the hand moves into the start position for the first time, data recording begins; henceforth, no data points are actually discarded (that is, if the data streams were appended they would represent the entirety of the block; the end of each data file is contiguous with the start of the next data file). Note that it is important to update the data path specified in config.h before each new block, or else data from several blocks of trials will be recorded in the same data folder, resulting in multiple files with the same trial number (but with different time stamps). It is helpful (but not entirely necessary) that the specified data path (folder) already exist.

Alternatively, the code may be designed such that data are recorded as a single continuous stream (that is, not parsed into individual trials). Although this method is less commonly applied, it allows for continuous and free recording throughout the entire session and is independent of the trial structure.

Data files generally contain the tracking data from the Ascension system (all connected trackers) or the mouse (1 tracker), along with any useful stimulus information. The contents of the data file can be modified by adjusting the information stored in the TargetFrame.h and InputFrame.h, and written out in DataWriter.cpp. Note that because of system delays and different clocks, the timestamp recorded in the data stream is an artificially generated timestamp corresponding to the time the data sample was recorded by the Ascension. In contrast, stimulus information is recorded at the time that the data sample is written out to the data file, which may occur after some arbitrary and variable temporal delay. Thus, while all attempts are made to keep the display, Flock of Birds, and data stream in sync, note that there are unavoidable timing uncertainties that currently cannot be avoided.

The header at the top of the data file contains information about the current time that the data file was opened, the sampling rate of the Ascension, and column names indicating what each column in the data steam represents. This data can be easily parsed using the KRload matlab function (written by Aaron Wong), which returns a data cell in which each cell corresponds to data from one tracker, and exists as a structure in which each column in the data file is parsed into a separate array. There is also an associated parameter file that contains the information stored in the header file.

**1.5 Connecting the Kinereach for the first time**

\*Note: This is often easiest to achieve by using a minimal debugging code external to Visual Studio whose sole job is to check the connection to the Flock of Birds or the TrakStar. Such a program, such as one compiled and executed on the command line, is available (ask Aaron Wong for more details).

Perhaps one of the most frustrating steps in setting up the Kinereach is getting the Flock of Birds Ascension hardware to interface with the computer. The following steps should ease this process. Note, these steps are only applicable to the Flock of Birds system. The TrakStar system should be much easier to connect, and does not require these steps.

For the Flock of Birds:

First install the USB to Serial driver. In general, this is the Tripp-Lite USA-49WG USB to Serial converter, whose driver can be found online or in the “Install Files” folder.

When the USB to Serial converter is plugged into the computer, 4 com ports should be present in the Device Manager (Start->Control Panel->Hardware and Sound->Device Manager->Ports). Open each com port (right click and choose Properties) and tab over to the “Advanced” section; then in the dropdown menu note the Physical Device Object Name to determine which serial port is being modified (i.e., the Device Name is the name of the particular serial connector on the converter). Then, set the com port number (Port Settings -> Advanced -> Com Port Number) as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Physical Device Object Name | Actual Device Serial Port Number | Com Port Number | Bird Number |
| 00 | Serial 1 | Com 5 | Bird 1 (left hand) |
| 01 | Serial 2 | Com 6 | Bird 2 |
| 02 | Serial 3 | Com 7 | Bird 3 (right hand) |
| 03 | Serial 4 | Com 8 | Bird 4 |

Also make sure that the com ports are set at a baud rate of 115200 (in Port Settings). Once these com ports are set, restart the computer.

In the C++ code (in TrackBird.h), make sure that the order of the com ports reflects the order of the com port numbers according to the Physical Device Object Name, e.g. {0, 5, 6, 7, 8}.

Once the com ports have been set up, the birds should all initialize properly.

For the TrakStar:

Connect the USB to the computer. No additional configuration of com ports is required.

**1.6 Calibrating the Kinereach**

Calibration of the Kinereach is quite a painful process, and is faciliated by use of the Calibration C++ Project code, the calibrateKR matlab script, and a plumb-bob.

To perform a new calibration, first remove the mirror, allowing direct line of sight between the display monitor and tracking birds. Run the calibration project. This will put up a display on the screen with 9 cross-marks (left figure):

1

6

7

2

5

8

3

4

9

The numbers in the right figure correspond to the order in which the cross-marks should be visited during the calibration. Suspend the plumb bob from a cross. Then, line up the left-hand or right-hand tracker on the bob, trying to be consistent about which side of the plumb bob the tracker resides (e.g., keep the left-hand tracker on the left side of the plumb bob). Leave the tracker in place while removing the plumb bob (to eliminate metal interference) for a few seconds to get a good measurement, then move on to the next cross-mark until all 9 have been visited. Repeat for at least two trackers. Although it is possible to calibrate more than one tracker at each point at the same time, note that allowing the trackers to come in very close proximity to each other will induce interference and throw off the calibration. Once at least two trackers have been calibrated, stop the recording.

In Matlab, calibrateKR and select the data file containing the calibration data. Follow the instructions to mark 9 good sections of data, one section (1 pair of points) for each target. Do this for both the left and right hand trackers. The program will then estimate an X and Y offset and a rotation angle (in radians). Copy these numbers into config.h; they will be used to translate raw Ascension data into calibrated position information.

For reference, note that the coordinate system of the Ascension may be rotated relative to the coordinate system of the display (note, this assumes the transmitter is aligned as in the original setup. If it is rotated, these axes must also be rotated in the TrackBird code). The systems are assumed to be oriented as such:

+X

+Y

+Z

transmitter

|  |  |
| --- | --- |
| Display | Birds |
| +X | +Y |
| +Y | +Z |
| +Z | +X |

display

+X

+Y

+Z

Note that this requires a change of axes, which is quite simple for describing translations but is more complex for describing angular rotations (and as of this writing has not been fully worked out, since the angular information is not used and because it depends upon the defined system used to determine the angles – recall, rotations are non-commutative. We avoid this problem in the new TrackBird version of the code by saving the rotation matrix instead of Euler angles). The change of axis for translations, as well as the calibration scaling and rotation parameters, are applied in the TrackBird object.

**2. Programming Manual**

**2.1 Configuration**

**2.1.1 Constants**

Configurable constants can be found in config.h. Below are a list of the main parameters to be adjusted in the config.h file.

SCREEN WIDTH, SCREEN HEIGHT

The resolution to use to run the application.

WINDOWED

If this value is true, the application runs in windowed mode. If it is false, the application runs in fullscreen mode.

MIRRORED

If this value is true, all OpenGL rendering is mirrored horizontally.

PHYSICAL WIDTH

The width of the screen in meters.

PHYSICAL HEIGHT

The height of the screen in meters.

TRACKTYPE

The type of tracker to be used. Set to 0 for the Flock of Birds, or 1 for the trackStar. It is critical that this parameter be set correctly or the program will fail to run in tracking mode.

FILTER\_WIDE

Sets the status of the wide-band hardware filter. Set to TRUE to engage, or FALSE to disengage. Default setting is TRUE.

FILTER\_NARROW

Sets the status of the narrow-band hardware filter. Set to TRUE to engage, or FALSE to disengage. Default setting is FALSE. In general, only the Wide or the Narrow filter should be on.

FILTER\_DC

Sets the status of the adaptive DC hardware filter. Set to 1.0f to engage, or 0.0f to disengage. It is possible to set this parameter to a value between 0.0 and 1.0 to change the settings of this filter (refer to Ascension manuals for details).

CALxOFFSET, CALyOFFSET, CALxyROTANG

The calibration constants for the horizontal and vertical offsets (in meters) and the angular rotation (in radians).

**2.1.2 COM ports**

This parameter has been moved to the TrackBird object.

The COM ports where the Flock of Birds are connected are defined as a WORD[] called COM port. The value at index 0 of this array should be 0, and all following values should be the COM port numbers of each of the birds of corresponding indices.

**2.2 Application (main.cpp)**

**2.2.1 Methods**

bool init();

Initializes everything. Loading of image, sound, and font files and initialization of pointers and cursors is done here. All images are loaded, and calles LoadTrTable to load the trial table.

This function calls on the TrackBird object to initialize the Ascention trackers. If TrackBird::InitializeBird returns a value less than 0, the initialization has failed; otherwise, it returns zero.

A number of other objects are also initialized here. These include the FTDI sensors, any specified Paths and Regions.

Returns true if there were no errors, false otherwise.

void setup opengl();

Sets up OpenGL. Also sets the color of the background.

void clean up();

Performs closing operations, including shutting down of the Ascension system via a call to the TrackBird object.

void draw\_screen();

Draws objects on the screen. Control is maintained by a drawstruc structure that includes flags to indicate what exactly is to be drawn.

void game update();

Main game loop (e.g., the state machine). Controls the logical flow of events that occur according to the experimental structure and the trial table parameters.

int LoadTrFile();

Loads a trial table by reading in the name of the trial table as defined in config.h (TRIALFILE), and stores the resulting table in an array of structures. Parameters (fields of the structure) from the current trial may be accessed by referring to curtr.

int main();

Controls the flow of the experiment; polls for updates from the Ascension and other input devices, and runs the game\_update and the draw\_screen functions.

**2.2.2 Loading files**

2.2.2.1 Objects

All pertinent object files (namely images, paths, and regions) can be loaded using the appropriate LoadFromFile() class function, which returns either a pointer to an object or the object itself.

2.2.2.2 Sounds

Sound files can be loaded by calling the Sound constructor and passing the file path. Note that for good timing control, the duration of the sound file should be limited to be quite short, as the sound buffer has been restricted in size.

2.2.2.3 Fonts

Font files can be loaded using the TTF OpenFont() function. The first parameter is the file path, and the second is the font size. The return value is a pointer of type TTF Font.

**2.2.3 Displaying objects**

All drawing should be done in the draw\_screen() function. All images can be displayed at given positions using the corresponding Draw() method for that class. In some cases (e.g., the Object2D class), the class can store position information, and its Draw() methods will use the appropriate position automatically. Note, in some cases the object draw status must first be set to On(), otherwise a call to Draw() will not render the desired image.

2.2.3.1 Text

To render text, use an SDL\_ttf render function [(see list)](http://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf.html#SEC42) to generate an SDL Surface. The color is specified by an SDL\_Color, which can be represented by an array containing red, green, and blue values (0.0 to 1.0). Use the SDL\_Surface to initialize an Image for the text, which can then be used like any other Image object.

Note, when creating text images for the screen it is important to delete a text value first rather than attempting to just write over it. Doing the latter will use up memory and eventually incur an out-of-memory error.

**2.2.4 Playing sounds**

To play a sound from a Sound object, simply call the Play() method when appropriate.

**2.2.5 Reading inputs**

2.2.5.1 TrackBird

To update inputs from the Ascension, data inputs are returned as an array of InputFrame structures (each element of the array corresponds to one tracker) following a call to TrackBird::GetUpdatedSample(). Data from each tracker may then be called individually by specifying the appropriate element of the array. Each element of the array is then passed to the DataWriter() function for output in the data file.

2.2.5.2 SDL Events

SDL events can be used to collect miscellaneous inputs such as mouse movement, joystick motion, and key presses. The best way to retrieve this data is by using the SDL\_PollEvent() function. This removes an event from the event queue and stores its data in the passed SDL\_Event parameter. Events with the SDL\_MOUSEMOTION type should be passed to the MouseInput::ProcessEvent() method to make the GetFrame() method return the most current data. As with TrackBird, the data is returned by updating the appropriate element (Element 0) in the data array of structures.

**2.3 Classes**

**2.3.1 InputFrame**

Data type used to transmit inputs. Consistent across all data types.

**2.3.2 TrackBird**

Handles all interface with an Ascension device (Flock of Birds or TrakStar).

2.3.2.1 InitializeBird

This function initializes the appropriate Ascension device according to the constant parameters specified in the config file.

int TrackBird:: InitializeBird(TrackSYSCONFIG \*sysconfig);

Returns a flag indicating successful initialization of the Ascension device (value = 0), or a code indicating the stage at which the initialization failed.

Parameter:

sysconfig

A structure of type TrackSYSCONFIG that specifies all of the system parameters.

2.3.2.2 GetUpdatedSample

This function contains all the logic for acquiring data from the ascension system and putting it into a calibrated format for use by the rest of the application.

int GetUpdatedSample(TrackSYSCONFIG \*sysconfig,TrackDATAFRAME dataframe[]);

Returns the number of samples acquired during the function call. This should either be 0 (indicating no new data has been acquired), or equal to the number of tracking sensors attached to the Ascension device.

Parameters:

sysconfig

A structure of type TrackSYSCONFIG that specifies all of the system parameters.

dataframe[]

The array of data frame structures which holds all of the newly acquired data from the sensors.

2.3.2.3 ShutDownBird

Shuts down the Ascension system appropriately.

int ShutDownBird(TrackSYSCONFIG \*sysconfig);

Returns a flag indicating success in shutting down the system (value = 0), or indicates the Ascension error code corresponding to the failure to shut down the tracker.

sysconfig

A structure of type TrackSYSCONFIG that specifies all of the system parameters.

**2.3.3 MouseInput**

Handles mouse actions. Inherits InputDevice.

2.3.3.1 GetFrame

int GetFrame(TrackDATAFRAME dataframe[]);

Returns the most recent frame of data from the mouse.

Parameter:

dataframe[]

The array of data frame structures which holds all of the newly acquired data.

2.3.3.2 ProcessEvent

Updates MouseInput with new position information.

static void ProcessEvent(SDL Event event);

Parameter:

event

An SDL Event containing the updated information.

**2.3.4 JoystickInput**

Handles mouse actions. Inherits InputDevice.

2.3.4.1 GetFrame

int GetFrame(TrackDATAFRAME dataframe[], int input);

Returns the most recent frame of data from the joystick.

Parameter:

dataframe []

The array of data frame structures which holds all of the newly acquired data.

input

The index into the dataframe array into which the joystick data will be saved.

2.3.4.2 ProcessEvent

Updates JoystickInput with new position information.

static void ProcessEvent(SDL Event event, SDL\_Joystick \*joystick);

Parameter:

event

An SDL Event containing the updated information.

joystick

Pointer to the joystick object.

2.3.4.3 CloseJoystick

Shuts down Joystick.

static void CloseJoystick(SDL\_Joystick \*joystick);

Parameter:

joystick

Pointer to the joystick object.

**2.3.5 DataWriter**

Records data.

2.3.5.1 Constructor

DataWriter(char\* filename = NULL);

Parameter:

filename

Custom filename. If not used, filename is generated from the date and time.

2.3.5.2 Record

Record a new line of data.

void Record(int deviceNo, TrackDATAFRAME frame, TargetFrame Target);

Parameters:

deviceNo

Number identifying the device sending data.

frame

Single InputFrame of data to be written out.

Target

TargetFrame with current trial parameters to be written out.

**2.3.6 Sound**

Stores a sound for playback.

2.3.6.1 Constructor

Sound(char\* filePath);

Parameter:

filePath

Path to the sound file.

2.3.6.2 GetChunk

Mix Chunk\* GetChunk() const;

Returns The SDL mixer chunk used to store the sound.

2.3.6.3 Play

Plays the sound.

void Play(int loops = 0);

Parameter:

loops

Number of times to replay the sound (default 0).

**2.3.7 Image**

Stores an image as an OpenGL texture and can draw it on-screen.

2.3.7.1 Constructor

Image(SDL Surface\* surface, float ratio = PHYSICAL RATIO);

Parameters:

surface

The SDL surface used to generate the image. Must not be a NULL pointer.

ratio

Resolution of the image in meters per pixel (defaults to global ratio).

2.3.7.2 LoadFromFile

Creates an Image object based on a file path.

static Image\* LoadFromFile(char\* filePath);

Parameter:

filePath

Path to the image file.

Returns a pointer to the new Image if it was created successfully. Otherwise, NULL.

2.3.7.3 GetTexture

GLuint GetTexture() const;

Returns the OpenGL texture for the image.

2.3.7.4 GetWidth

GLfloat GetWidth() const;

Returns the width (meters) of the image.

2.3.7.5 GetHeight

GLfloat GetHeight() const;

Returns the height (meters) of the image.

2.3.7.6 Draw

Draws the image in a specified position.

void Draw(GLfloat xPos, GLfloat yPos, GLfloat theta = 0.0f);

void Draw(GLfloat xPos, GLfloat yPos, GLfloat w, GLfloat h, GLfloat theta = 0.0f);

Parameters:

xPos

The X position(meters) of the image.

yPos

The Y position (meters) of the image.

w

The width (meters) to use for drawing the image.

h

The height (meters) to use for drawing the image.

theta

The counterclockwise rotation (radians) of the image (default 0).

2.3.7.6 DrawAlign

Draws the image in a specified position, with the alignment specified with a flag.

void DrawAlign(GLfloat xPos, GLfloat yPos, GLfloat w, GLfloat h, GLint cflag = 0);

Parameters:

xPos

The X position(meters) of the image.

yPos

The Y position (meters) of the image.

w

The width (meters) to use for drawing the image.

h

The height (meters) to use for drawing the image.

cflag

A flag indicating how to align the object (that is, which side of the object is specified by the xPos and yPos parameters. Alignment is: 1 = Right, 2 = Up, 3 = Left, 4 = Down; 0 (default) is centered.

**2.3.8 Object2D**

Stores an image along with position and rotation information.

2.3.8.1 Constructor

Object2D(Image\* i);

Parameter:

i

A pointer to the Image representing this object.

2.3.8.2 GetWidth

GLfloat GetWidth() const;

Returns the default width (meters) of the image.

2.3.8.3 GetHeight

GLfloat GetHeight() const;

Returns the default height (meters) of the image.

2.3.8.4 GetX

GLfloat GetX() const;

Returns the X position of the object.

2.3.8.5 GetY

GLfloat GetY() const;

Returns the Y position of the object.

2.3.8.6 GetAngle

GLfloat GetAngle() const;

Returns the counterclockwise rotation (radians) of the object.

2.3.8.7 SetPos

Sets the position (meters) of the object.

void SetPos(GLfloat x, GLfloat y);

Parameters:

x

The X position to set.

y

The Y position to set.

2.3.8.8 SetAngle

Sets the counterclockwise rotation (radians) of the object.

void SetAngle(GLfloat theta);

Parameter:

theta

The angle of rotation to set.

2.3.8.9 Draw

Draws the object.

void Draw();

void Draw(GLfloat width, GLfloat height);

Parameters:

width

The width (meters) to use for drawing the object.

height

The height (meters) to use for drawing the object.

2.3.8.10 Distance

Gets the distance between two objects (meters).

float Distance(Object2D\* ob1, Object2D\* ob2);

float Distance(Object2D\* ob1, GLfloat x, GLfloat y);

Parameters:

ob1, ob2

Pointers to the objects to compare.

x

The X position of the second object.

y

The Y position of the second object.

2.3.8.11 On

Set the draw state to be on.

void On()

2.3.8.11 Off

Set the draw state to be off.

void Off()

**2.3.9 Region2D**

Defines a polygonal region of the screen.

2.3.9.1 SetNSides

Sets the number of sides of the polygon (minimum of 3, maximum of 10)

void SetNSides(GLint sides);

Parameter:

sides

Number of sides of the polygonal region.

2.3.9.2 SetPolyVerts

Defines the vertices of polygon according to the number of sides of the polygon.

void SetPolyVerts(GLfloat Verts[][2])

Parameter:

Verts

Array containing the (x,y) vertices of the polygon.

2.3.9.3 SetOneVert

Redefines one value of one vertex of polygon.

void SetOneVert(GLint i, GLint j, GLfloat vert)

Parameter:

i, j

The index of the vertex value to be modified (e.g., i is the vertex number and j identifies whether it is the x or y position to be adjusted).

vert

The new x or y vertex value to be stored.

2.3.9.4 SetPolyColor

Sets the color of polygon.

void SetPolycolor(GLfloat clr[])

Parameter:

clr[]

An array containing the [red,green,blue] value defining the desired color.

2.3.9.5 GetPolyVert

Retrieve one value of one vertex of the polygon.

GLfloat GetPolyVert(GLint i, GLint j)

Parameter:

i, j

The index of the vertex value to be retrieved (e.g., i is the vertex number and j identifies whether it is the x or y position).

2.3.9.6 GetPolySides

GLint GetPolySides()

Returns the number of sides of the polygon

2.3.9.7 LoadPolyFromFile

Reads in a specified file and returns a completely defined Region2D object. The file structure should contain the polygon color on the first line, and then each subsequent line should contain one pair of vertices. The number of sides (up to 10) will be determined from the number of vertices existing in the file.

Region2D LoadPolyFromFile(char\* filePath)

Parameter:

filePath

The path of the region file to be read in.

2.3.9.8 Draw

Draws the region.

void Draw();

2.3.9.9 InRegion

Determines if the object or cursor is within the polygon.

bool InRegion(Object2D\* cursor)

bool InRegion(HandCursor\* cursor)

bool InRegion(GLfloat xcurs, GLfloat ycurs)

Parameters:

cursor

Pointer to an Object2D or HandCursor object whose position will be referenced relative to the region.

xcurs, ycurs

The (x,y) position; it is to be determined whether that position resides inside the region.

Returns true if inside the region, otherwise returns false.

**2.3.10 Path2D**

Defines a path (line, boundary, barrier, etc) in the workspace. Segments may be comprised of straight lines or circular arcs.

2.3.10.1 SetNVerts

Sets the number of line segments of the path (minimum of 1, maximum of 8)

void SetNVerts(GLint sides);

Parameter:

sides

Number of vertices in the path.

2.3.10.2 SetPathVerts

Defines the vertices of the segments path.

void SetPolyVerts(GLfloat Verts[][6])

Parameter:

Verts

Array defining the segments of the path.

For straight line segments, the first two columns of the array contain the horizontal and vertical position of the first vertex, and the second two columns define the horizontal and vertical positions of the second vertex. The remaining columns should be zero.

For circular arc segments, the first two columns define the center horizontal and vertical positions of the arc; the third column is the radius, the fourth column is the start angle (radians) and the fifth column is the arc length (radians). The last column is set to 1 to flag the segment as a circular arc.

2.3.10.3 SetOneVert

Redefines one value of one segment of the path.

void SetOneVert(GLint i, GLint j, GLfloat vert)

Parameter:

i, j

The index of the value to be modified (e.g., i is the segment number and j identifies the parameter value of that segment).

vert

The new vertex value to be stored.

2.3.10.4 SetPathColor

Sets the color of path.

void SetPathColor (GLfloat clr[])

Parameter:

clr[]

An array containing the [red,green,blue] value defining the desired color.

2.3.10.5 SetPathWidth

Sets the width of the line to be drawn.

void SetPathWidth(GLfloat width)

Parameter:

width

A value representing the width of the line to be drawn.

2.3.10.6 GetPathVert

Retrieve one value of one vertex of the path.

GLfloat GetPathVert(GLint i, GLint j)

Parameter:

i, j

The index of the vertex value to be retrieved (e.g., i is the vertex number and j identifies whether it is the x or y position).

2.3.10.7 GetPathNVerts

GLint GetPathNVerts ()

Returns the number of segments of the path.

2.3.10.8 LoadPathFromFile

Reads in a specified file and returns a completely defined Path2D object. The file structure should contain the path color on the first line, the path width on the second line, and then each subsequent line should contain one path segment. The number of segments (up to 8) will be determined from the number of vertices existing in the file.

Path2D LoadPathFromFile(char\* filePath)

Parameter:

filePath

The path of the region file to be read in.

2.3.10.9 Draw

Draws the region at the specified position.

void Draw(Glfloat pathx, GLfloat pathy);

Parameters:

pathx,pathy

The horizontal and vertical positions defining the origin from which the path vertices will be defined.

2.3.10.10 OnPath

Determines if the object is on the path.

bool OnPath(Object2D\* cursor,GLfloat pathx, GLfloat pathy);

bool OnPath(HandCursor\* cursor,GLfloat pathx, GLfloat pathy);

bool OnPath(float xcurs, float ycurs, GLfloat pathx, GLfloat pathy);

Parameters:

cursor

Pointer to an Object2D object whose position will be referenced relative to the path.

pathx, pathy

The horizontal and vertical positions defining the origin from which the path vertices will be defined.

Returns true if the cursor is intersecting the path, otherwise returns false.

2.3.10.11 PathCollision

Determines if the object has collided with the path (e.g., whether it’s trajectory has crossed the path).

bool PathCollision(Object2D\* cursor, Glfloat pathx, GLfloat pathy, Object2D\* LastCursorPos)

Parameters:

cursor

Pointer to an Object2D object whose position will be referenced relative to the path. This in general is the current position of the cursor.

pathx, pathy

The horizontal and vertical positions defining the origin from which the path vertices will be defined.

LastCursorPos

Pointer to an Object2D object that stores the previous position of the cursor one time step ago; this will be used in conjunction with the cursor parameter to define the cursor trajectory and determine if the cursor has moved across any portion of the path.

Returns true if the cursor trajectory has crossed the path, otherwise returns false.

2.3.10.12 HitViaPts

Determines if the object exists on a vertex (i.e., an end of a line segment).

int HitViaPts(Object2D\* cursor,GLfloat pathx, GLfloat pathy, GLfloat dist);

int HitViaPts(HandCursor\* cursor,GLfloat pathx, GLfloat pathy, GLfloat dist);

int HitViaPts(float xcurs, float ycurs, GLfloat pathx, GLfloat pathy, GLfloat dist);

Parameters:

cursor

Pointer to an Object2D object whose position will be referenced relative to the path. This in general is the current position of the cursor.

pathx, pathy

The horizontal and vertical positions defining the origin from which the path vertices will be defined.

If the cursor is on a vertex, returns the vertex number; otherwise, returns -99.

**2.3.11 Circle**

Draws a circle. Also serves as the basis for the HandCursor class.

3.3.11.1 Constructor

Circle(GLfloat x, GLfloat y, GLfloat diam, GLfloat clr[]);

Parameter:

x,y

The horizontal and vertical position of the center of the circle, in meters.

diam

The diameter of the circle, in meters.

clr[]

An array containing the [red,green,blue] value defining the desired color.

2.3.11.2 SetDiameter

Sets the diameter of the circle.

void SetDiameter(GLfloat diam);

Parameter:

diam

The diameter of the circle, in meters.

2.3.11.3 SetRadius

Sets the radius of the circle.

void SetRadius(GLfloat rad)

Parameter:

rad

The radius of the circle, in meters.

2.3.11.4 MakeVerts

Generates the vertices defining the boundary of the circle.

void MakeVerts();

2.3.11.5 SetColor

Sets the color of circle fill.

void SetColor (GLfloat clr[])

Parameter:

clr[]

An array containing the [red,green,blue] value defining the desired color.

2.3.11.6 SetBorderColor

Sets the color of border of the circle.

void SetBorderColor (GLfloat clr[])

Parameter:

clr[]

An array containing the [red,green,blue] value defining the desired color.

2.3.11.7 SetPos

Sets the horizontal and vertical position of the center of the circle.

void SetPos(GLfloat x, GLfloat y)

Parameter:

x,y

The horizontal and vertical position of the center of the circle, in meters.

2.3.11.8 setBorderWidth

Sets the width of the border.

void setBorderWidth(GLfloat w)

Parameter:

w

The border of the circle, in meters.

2.3.11.9 Draw

Draws the circle.

void Draw();

2.3.11.10 GetX

GLfloat GetX()

Returns the horizontal position of the center of the circle.

2.3.11.11 GetY

GLfloat GetY()

Returns the vertical position of the center of the circle.

2.3.11.10 GetDiam

GLfloat GetDiam()

Returns the diameter of the circle.

2.3.11.11 GetRadius

GLfloat GetRadius()

Returns the radius of the circle.

2.3.11.12 drawState

GLint drawState()

Returns the drawing state of the circle (1 means the circle will be drawn, 0 means it will not).

2.3.11.13 Distance

Calculates the distance between the circle and another object (meters).

GLfloat Distance(Circle\* c)

GLfloat Distance(Object2D\* c)

Parameters:

c

Pointer to the object (Circle or Object2D) whose distance from the circle will be computed.

2.3.11.14 On

Set the draw state to be on.

void On()

2.3.11.15 Off

Set the draw state to be off.

void Off()

2.3.11.16 BorderOn

Set the draw state of the border to be on.

void BorderOn()

2.3.11.15 BorderOff

Set the draw state of the border to be off.

void BorderOff()

**2.3.12 HandCursor**

Defines a circle as a HandCursor class, and imbues it with additional properties.

2.3.12.1 Constructor

HandCursor(Circle\* c);

Parameter:

c

Pointer to an existing circle that will be designated as a cursor object.

2.3.12.2 UpdatePos

Updates the position of the hand cursor. This updates both the actual location according to the tracker, and also the displayed position of the circle on the screen (which are separate). Also computes the instantaneous velocity of the tracker (note, the velocity of the displayed circle is not computed).

void UpdatePos(GLfloat x, GLfloat y);

Parameter:

x,y

The horizontal and vertical position of the actual tracker location, in meters.

2.3.12.3 SetRotation

Sets the angle of the visual rotation of the cursor (e.g., it defines the rotation matrix).

void SetRotation(GLfloat theta)

Parameter:

theta

The rotation angle of the cursor, in degrees.

2.3.12.4 SetOrigin

Defines the origin of the coordinate system about which rotations of the cursor occur.

void SetOrigin(GLfloat x, GLfloat y);

x,y

The horizontal and vertical position of the origin (center of rotation), in meters.

2.3.12.5 Null

Resets the cursor rotation to be zero.

void Null()

2.3.12.6 SetClamp

Sets the rotation angle along which the cursor motion is clamped (e.g., only radial distance reflects the true extent of the tracker away from the origin).

void SetClamp(GLfloat th)

Parameter:

th

The rotation angle along which cursor motion is clamped.

2.3.12.7 SetGain

Sets the gain on the cursor.

void SetGain(GLfloat xg, GLfloat yg)

Parameters:

xg,yg

The horizontal and vertical gains used to scale the cursor motion.

2.3.12.8 SetColor

Pass through function to set the fill color of the underlying circle object.

void SetColor(GLfloat clr[])

Parameter:

clr[]

An array containing the [red,green,blue] value defining the desired color.

2.3.12.9 Draw

Pass-through function controlling the drawing of the underlying circle object.

void Draw();

2.3.12.10 GetX

GLfloat GetX()

Returns the horizontal position of the center of the underlying circle object.

2.3.12.11 GetY

GLfloat GetY()

Returns the vertical position of the center of the underlying circle object.

2.3.12.12 GetTrueX

GLfloat GetTrueX()

Returns the horizontal position of the tracker location, in meters.

2.3.12.13 GetTrueY

GLfloat GetTrueY()

Returns the vertical position of the tracker location, in meters.

2.3.12.14 GetXVel

GLfloat GetXVel()

Returns the horizontal velocity of the tracker, in meters/second.

2.3.12.15 GetYVel

GLfloat GetYVel()

Returns the vertical velocity of the tracker, in meters/second.

2.3.12.16 GetVel

GLfloat GetVel()

Returns the absolute velocity of the tracker (e.g. the magnitude of the velocity vector), in meters/second.

2.3.12.17 Distance

Calculates the distance between the hand cursor and another object (meters).

GLfloat Distance(Circle\* c)

GLfloat Distance(Object2D\* c)

GLfloat Distance(GLfloat x, GLfloat y)

Parameters:

c

Pointer to the object (Circle or Object2D) whose distance from the circle will be computed.

x,y

Horizontal and vertical position from which the distance to the cursor is to be computed.

2.3.12.18 On

Set the draw state of the underlying circle object to be on.

void On()

2.3.12.19 Off

Set the draw state of the underlying circle object to be on.

void Off()

2.3.12.20 SetHitMargin

Sets the fudge-factor margin of error surrounding the target that will still be considered as a “hit”.

void SetHitMargin(GLfloat m)

Parameter:

m

The multiplicative factor for the number of radii that will be considered as a hit.

2.3.12.20 GetHitMargin

GLfloat GetHitMargin()

Returns the value of the hit margin, the fudge-factor margin of error surrounding the target that will still be considered as a “hit”.

2.3.12.21 HitTarget

Determines if the cursor has hit the target (or has ever hit the target within the last few samples).

bool HitTarget(Circle\* targ)

bool HitTarget(GLfloat x, GLfloat y, GLfloat rad)

Parameters:

targ

Pointer to a circle object that is the target whose hit status will be determined.

x,y

Horizontal and vertical positions defining the center of a target whose hit status will be determined.

rad

Radius around the center of the target that will be used to determine the region in which a “hit” is defined.

Returns true if the hand cursor has intersected the target recently, otherwise returns false.

**2.3.13 Ftdi**

Handles all interface with the FTDI digital I/O cable.

2.3.2.1 InitFtdi

This function initializes the FTDI device(s).

int InitFtdi(int i, FT\_HANDLE \*ftHandle, int devMode = 1, UCHAR mask = 0x00);

Returns a flag indicating successful initialization of the FTDI device (value = 0), or a code indicating the stage at which the initialization failed.

Parameter:

ftHandle

Pointer to an FTDI device handle.

devMode

Flag indicating how to initialize the FTDI. The default is mode 1, which is the bitbang mode. If this flag is set to anything other than 1, the FTDI will be set to serial mode.

mask

For BitBang mode. The mask indicates how each line should be initialized (i.e., as input or output). Set the bits in the upper nibble of the mask to 1 to be output lines, and 0 to be input lines. The lower nibble states whether the output lines should be set high or low. If the line is an input line, the lower nibble should set the line to 1 to allow inputs to pass freely through.

2.3.2.2 GetFtdi

This function reads the FTDI data lines, in serial mode. Function has not been debugged to ensure proper functioning.

int GetFtdiData(FT\_HANDLE ftHandle)

Returns a flag indicating the current status of the CTS line (0 or 1), or -1 if failed to read.

Parameters:

ftHandle

Pointer to an FTDI device handle.

2.3.2.3 GetFtdiBitBang

This function reads the value of an FTDI data line, in BitBang mode, and returns the value of the bit on the specified line.

int GetFtdiBitBang(FT\_HANDLE ftHandle, int bit)

Returns the current status of the specified line (0 or 1), or a status flag (negative number) if it failed to read.

Parameters:

ftHandle

Pointer to an FTDI device handle.

bit

Flag indicating which line to read data from. The default is to read data off the CTS line, which is bit 4.

2.3.2.3 SetFtdiBitBang

This function sets the value of an FTDI data line, in BitBang mode, if that line has been set to an output line.

int SetFtdiBitBang(FT\_HANDLE ftHandle,UCHAR mask, int bit, int value)

Returns a flag indicating success in setting the specified line (0), or a negative value if the attempt has failed (-1 meaning the requested line is not an output line).

Parameters:

ftHandle

Pointer to an FTDI device handle.

mask

The current mask status; the upper nibble indicates which lines are input and output lines. The lower nibble is ignored; this part of the mask (denoting status of the input and output lines) is determined from the current values of the I/O lines.

bit

Flag indicating which line to set the value of.

value

The value to set on the specified line (0 or 1).

2.3.2.3 CloseFtdi

Shuts down the Ascension system appropriately.

int CloseFtdi(FT\_HANDLE ftHandle, int devMode = 1)

Returns a flag indicating success in shutting down the system (value = 0), or indicates the Ascension error code corresponding to the failure to shut down the tracker.

Parameters:

ftHandle

Pointer to an FTDI device handle.

devMode

Flag indicating how the FTDI was initialized. The default is mode 1, which is the bitbang mode. This ensures that the FTDI is properly reset to its default state before it is shut down.