Group 4: Enhanced Gaming and Pointing

Students: David Dryjanski and Andrew Pinkham
TA: Chris Forker

6.111 Introductory Digital Systems Laboratory
Spring 2005

5/12/2005

Abstract

The scope of our project is to use head-mounted accelerometers and a camera to create a more natural gaming interface for the Xbox and to create an intuitive computer interface to allow disabled people to use their computer. The digital system reads conversion values from A/D converters that sample the accelerometer outputs, computes tilt angles and extracts other information from the accelerometers, and uses D/A converters to control a hacked Xbox controller. The data is also correlated to movement for a computer mouse and the system is interfaced to a hacked mouse. The video portion takes in data from the video camera, extracts the necessary information, analyzes the data, and maps the information into a position reference.

Introduction

Our project will provide 5 degree-of-freedom sensing (3 axis for head, 2 for body movement) input for Xbox games and computer pointing. For head tilt, we will use an accelerometer. The accelerometer will also respond to very high acceleration as a left click on a mouse button. For movement in game play we will use a camera to detect body motion. This will interface through an Xbox controller, and for a computer mouse through a PS/2 port. Another accelerometer detects jumping and crouching in the Xbox game.

Design Methodology / Design Tradeoffs

Design Methodology/Design Tradeoffs (Overall):

Aiming is controlled by the accelerometer headset because it seemed intuitive to aim by moving your head. The overhead camera is used to control movement because it is more reliable than accelerometers. Accelerometers would have required double integration to get movement, and there would have to be some feedback, such as a button press that would recalibrate the system. Due to these tradeoffs we implement with the video camera and accelerometer setup.

Design Methodology/Design Tradeoffs (Video):

The premise behind the video interface methodology was to allow anyone to play without any special form of equipment. This would make it easier for anyone to just pick up and play. In order to due this, all the color data was ignored, and the interface simply analyzed the luminance data. Acclimating to the 6.111 Lab environment, we chose to detect people as darker objects to contrast against the bright, white floor. In order to ensure proper edge detection, the surrounding environment had to be lighter than the person playing the game. This required covering the Xbox controller cable with masking tape and blurring the video image to reduce the dark tile-lines on the floor. Since video quality was not a priority, we ignored most of the video data and only concentrated on the odd active field data. This simplified the system tremendously and was accurate enough to meet the specifications of the system. The overall implementation worked well with our design methodology in ensuring quality body-movement recognition and gameplay.

Design Decisions/Methodology (Headset):

We decided to implement the headset using accelerometers because they were small, lightweight, ultra low-power, accurate enough for our application, had a high sampling frequency. and they were free samples from Analog Devices. We had to choose whether the accelerometers were going to measure dynamic acceleration or tilt. Measuring the accelerations due to movement would make measuring rotation easy. Though, keeping track of the absolute position would be difficult because the double integration of the acceleration and A/D conversions introduce too much error. Without any other kind of correction or feedback, it would have been impossible to keep the aiming calibrated. Therefore we decided to Use the accelerometers to measure tilt. It is easy to convert the axis components of the accelerometer to X and Y tilt angles and this method is able to accurately calibrate the system without any form of feedback. We also added a vertical accelerometer to detect jumping and crouching (note: this wasn't demonstrated because the controller got fried due to a short in the soldered USB cable connection).

Design Methodology/Design Tradeoffs (Xbox and Mouse Conversions):

Due to the difficulty in reverse engineering Microsoft's USB protocol, we hacked the Xbox controller interface instead. We also performed the same operation with the Microsoft IntelliMouse, which turned out to be a much more difficult hack.

Headset Interface

The function of the headset is to allow users to aim in Xbox games and control their computer pointer by tilting and rotating their head. A strap fits over the top of the users head. The ADXL203EB 2-axis accelerometer is mounted flat on top of the headset. Full-scale controller input is designed to occur at an angle of +/- 30 degrees from the normal in both dimensions. Tilting your head 30 degrees to the left, right, up, or down from its upright position corresponds to maxing out the Xbox analog stick. To convert from the ADXL203 signal to a linear movement input for the controller, an inverse sine lookup table is used. The ADXL203 voltage must go into 5/6 voltage divider before being input to the AD670. This is done for both accelerometer axes. There is a button on the side of the headset to effectively zero both accelerometer axes to adjust for slight misalignments of the headset. To make aiming more realistic, a compass sensor is also mounted on top of the headset. The sensor outputs a sin and cos curve which are converted by AD670. The compass algorithm works by assuming sin is piecewise linear from 0 until it equals cos, and then cos is linear until sin crosses the voltage at which sin and cos are equal, and then sin is linear until it crosses cos again, etc. The algorithm divides the signal into four sections, in each either sin or cos is approximately linear, and then can calculate the heading from that linearized signal. This input is added to the left - right aiming accelerometer axis, so that either rotation by up to 30 degrees or tilt by up to 30 degrees looks left or right. The computed values are output to the AD558 or other DAC. There are 2 potentiometers for the aiming stick on the controller, one for each axis, which are normally turned by the analog stick. These are replaced by the signal from the DAC. If the DAC voltage is properly calibrated, the controller will respond exactly as if the potentiometers were still in place and the analog stick was being used.

The headset can also be used as a mouse to control the pointer, or can be used as a computer game controller. To allow the user to make fine adjustments with the mouse while still using the headset, both can be used at the same time. The computed values for both axes are converted to speed, distance, and direction parameters to the mouse control module. The distance and direction values are used to compute the number of pixels to move in each direction. The speed determines the period between the pulses which simulate mouse movement. The width of the pulse is fixed and is dependent on the mouse that is used. There are two pulses for each mouse axis. Depending on which pulse goes low first, the mouse will move in a different direction in that axis. The mouse is signaled to click when the user quickly moves their head down or up. This generates an acceleration greater than 1g or less than -1g. If the accel module detects an input greater than 1g, it takes a signal low which drives the left button pin low, and if the input is less than -1g, it takes the right click button pin low. This allows people who have difficulty using a mouse effectively to control the pointer by simply looking at a point on the screen. Because it calculates absolute position on the screen from tilt and rotation angles of the head and not from dynamic accelerations, even if the users head is not perfectly stable, the pointer should maintain a stable position.

Video Interface

The objective of the video interface module is to correlate a player's body movement into Xbox controller movement signals. The module must correctly analyze incoming data from a video camera into a digital direction vector. In order to emulate a joystick, a camera will be suspended from the ceiling focusing on a light-colored square shaped region and a player, wearing dark colors, will stand in the center of the region. Movement will be determined by the player's position relative to the center. The entire module is divided into three main sub-modules: a sub-module dedicated to locating essential data being received from the video decoder, another sub-module that interprets the data to determine the position of the player, and a final sub-module that converts the position of the into a direction vector (i.e. angle and magnitude). The module consists of a 7185 decoder and a Video Top Module that instantiates each of the Minor FSM sub-modules: Frame-Finder, Edge-Detector, and Mapper. It begins by enabling the Frame-Finder to locate the active video region, then enables the Edge-Detector to locate the position of the person, and finally enables the Mapper FSM to convert the position into a signal that can be fed into the Xbox.

The Frame-Finder FSM is controlled by the Major FSM and takes the decoded video signal as input and outputs video data within the active video region. This is done by sampling the incoming data and looking for the Start of Active Video (SAV) and End of Active Video (EAV) signals. The FSM waits for the 3FF, 000, and 000 samples and then checks the XYZ word to determine whether the data is SAV or EAV. If XYZ word is SAV for the odd active video region (i.e. 10'h200), the FSM begins to output the video data, until it reaches another 3FF signal. If the signal is EAV, the FSM waits again for the SAV signal to start outputting data. Once the module detects the 10'h2D8 time code, which signals the end of odd active video, the module resets the line count and waits until the start of the next odd field. Furthermore, Frame-Finder outputs signals start_odd_load and to signal to the Edge_Detect FSM when the data should be analyzed.

The Edge-Detector detects regions of contrast to determine where the person is located on the mat. This is accomplished by sampling the pixels in an entire frame and comparing their luminance values to a threshold luminance. For each pixel with enough luminance, the position of that pixel is compared against the current position of the most extreme pixels calculated thus far: x_min, x_max, y_min, and y_max. The position is determined through counters that count each vertical and horizontal line. In this manner the entire frame is analyzed until the most extreme positions of the person remain. This gives an overall area of the person which feeds into the Mapper FSM to calculate the actual position.

The Mapper FSM uses the extreme position data and references it to the mat area. The predefined parameters for the center of the mat (y_mid and x_mid) and for left, right, up, and down boundaries (L, R, U, D), are used to measure the relevant position of the person. Using a relatively basic algorithm, the direction vector can be calculated by comparing distances from the center and from the boundaries. For a standard 9-direction square pad, the person's direction can be calculated by averaging the extreme points (x_min, x_max, y_min, y_max) to find the centroid of the person (x_crit, y_crit). The (x_crit, y_crit) point is then mapped to one of the 9 direction boxes. The magnitude can be calculated from the distance of the person from the boundary of the box, although we only cared about the discrete position of the person.

Xbox and Mouse Conversion Interface

For the Xbox Controller, each analog stick is implemented using 2 potentiometers for each axis. We discovered that it was possible to replace the middle node of the pot with a DAC output between .1 and 3.5V. This correlated well with the accelerometers that had nearly the full range with 150Kohms. The right and left triggers also used the same potentiometer implementation.

The mouse interface was much more complicated. The ball in the mouse rotates two rollers and wheels with grooves cut in it. There is an infrared LED that is always on one side, and a phototransistor pair on the opposite side. One of them is darkened first depending on the direction the ball is rolling. Then both turn off and on. We replaced the phototransistors with current sources controlled by the digital system. The timing of the active low dark pulses is critical and kept fixed. Since there was no datasheet for the chip in the mouse that takes as input the phototransistor pulses, we continued to tweak the system until it worked properly. The period between these pulses characterizes the speed of the mouse and is generally lasts a few milliseconds. Clicking was also implemented by a single BJT on the mouse button whose gate was biased by a digital output. When the vertical accelerometer detects greater than about 1.2g (jumping right when your legs spring you off the ground) the jump button is pulsed and crouching is disabled so a crouch is not triggered when you are descending. When it detects less than about 1/2g, the crouch button is held low until a jump is detected.

Testing/Simulations

The video system took up the most significant portion of testing and simulation. Due to the difficulty in generating video data for valid simulations, most of the video testing was performed through the logic analyzer. One of the major realizations for the video interface was that the line and sample counters were sometimes inaccurate. This led us to focus mainly on the video time codes embedded within the data stream to discern one line from another. Another major issue for video was timing synchronization. Since the video decoder clock differed slightly from the overall system clock, the video module had to be completely separate from the rest of the project. Glitches in the incoming video data and a semi-functional logic analyzer also added to complexity in having a fully functional system. As for testing the actual position mapping with people went smoothly and just required some fine tuning to get the optimal directional grid.

Another strange malfunction occurred during the first take of the project demonstration. Even though the project was in working order 2 minutes before the presentation, when filming began, the Xbox console did not recognize our hacked Xbox controller. Having no time to discover the cause of the problem, we quickly reassembled a new controller that worked flawlessly in the second take of our presentation.

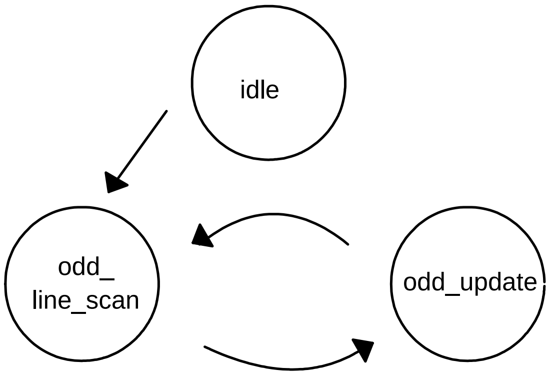
Once we learned that accelerometers could be applicable for this project idea, we acquired them and began testing. We had trouble getting good data when I used the accelerometers with the AD670 because of loading causing voltage drop. The error was consistent, so we were able to test around it and accurately measure acceleration components. We then created a ROM module that contained 120 computed values of arcsin to calculate angles and verified the ROM and its control FSM in simulation. This just made angle proportional to how much the aim moves. In practice, this wasn’t very desirable for aiming because it required too much tilt for small adjustments while not allowing a large enough range. We then calculated the output voltages directly from the A/D values which was better because it allowed you to aim without tilting as much.

To test the Xbox hack, we measured the voltage as the analog stick moved and determined which pads on the circuit board needed to be connected to. We then removed the potentiometers and soldered to the PCB pads. We then verified the idea first by connecting it to a variable voltage power supply, connecting the USB cable to a computer, and used the PC drivers to test the controller. We approached the hacked mouse in a similar manner ensuring that the testing emulated the phototransistor’s behavior.

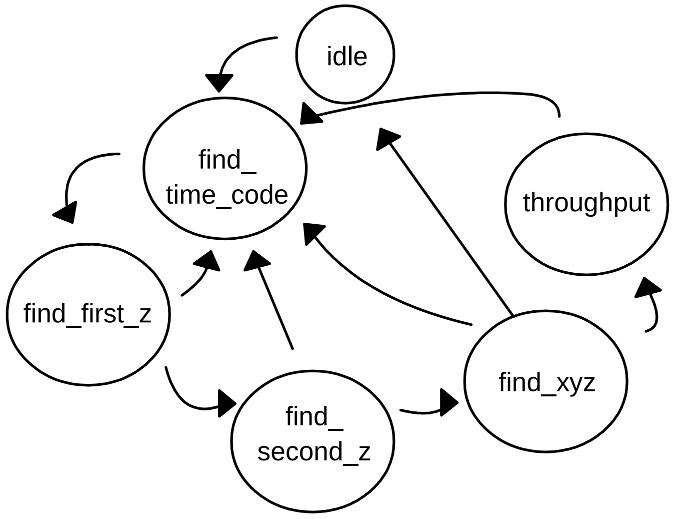
Conclusions

The final project was a great overall experience. Although taking up a huge amount of time, we knew the long-term benefits were worth it. Once the system became fully functional, it was great just being able to actually play video games on our virtual reality system. We only wish we could have had more time to finish implementing a jump and crouch function with another accelerometer.

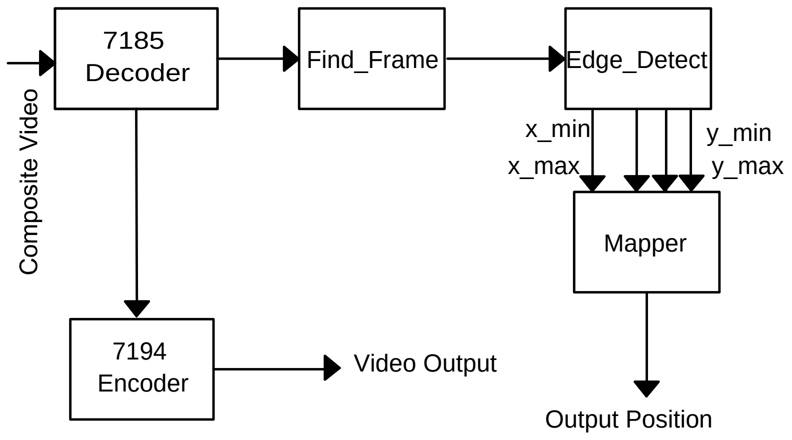
Edge-Detect FSM



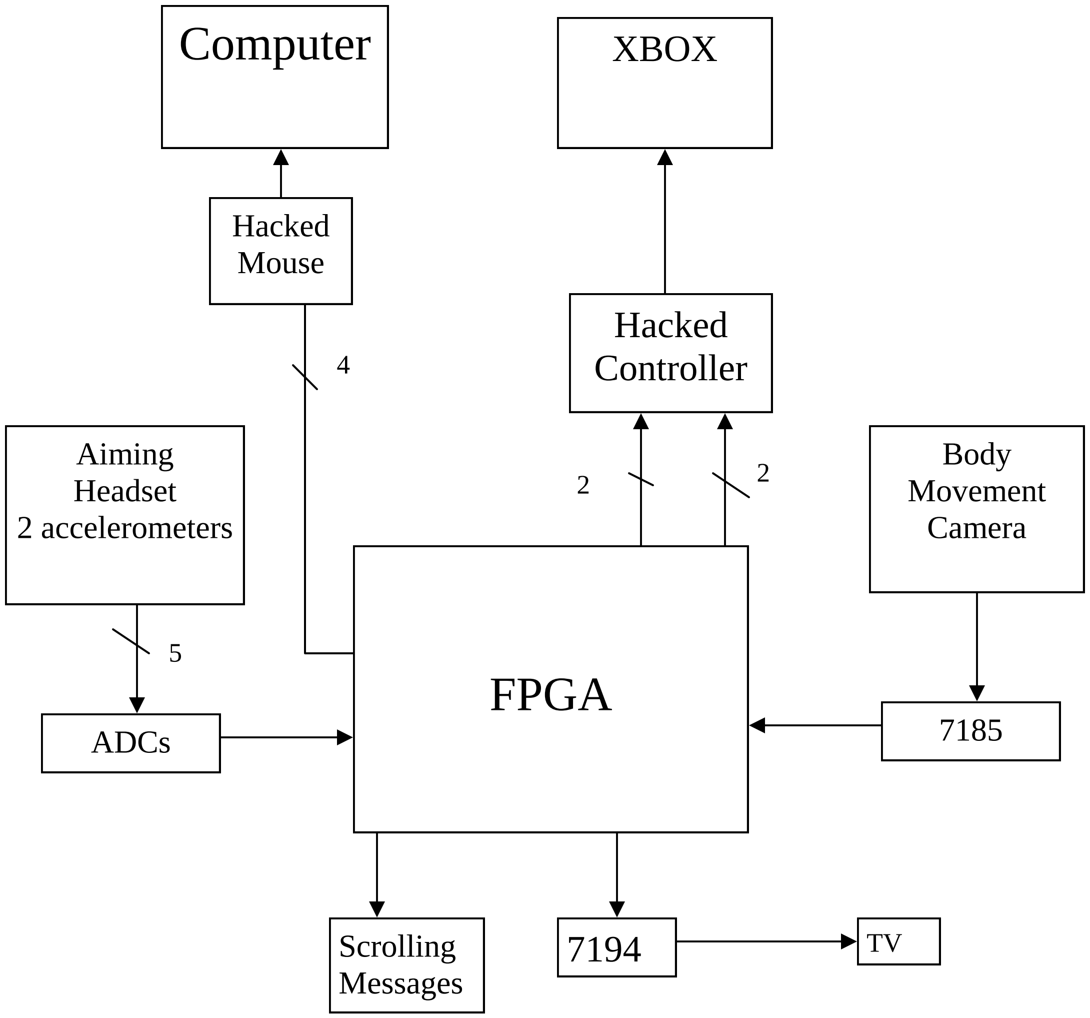
Find-Frame FSM

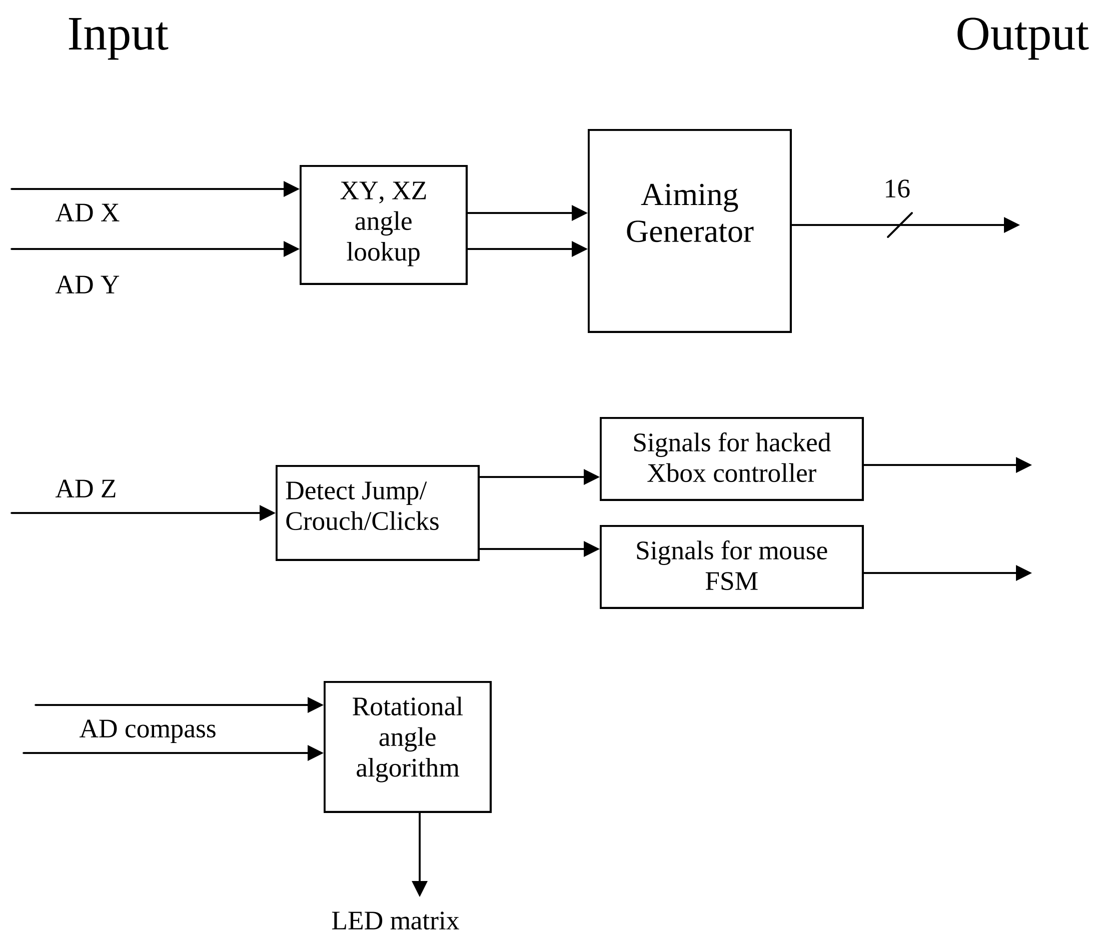


Video Block Diagram

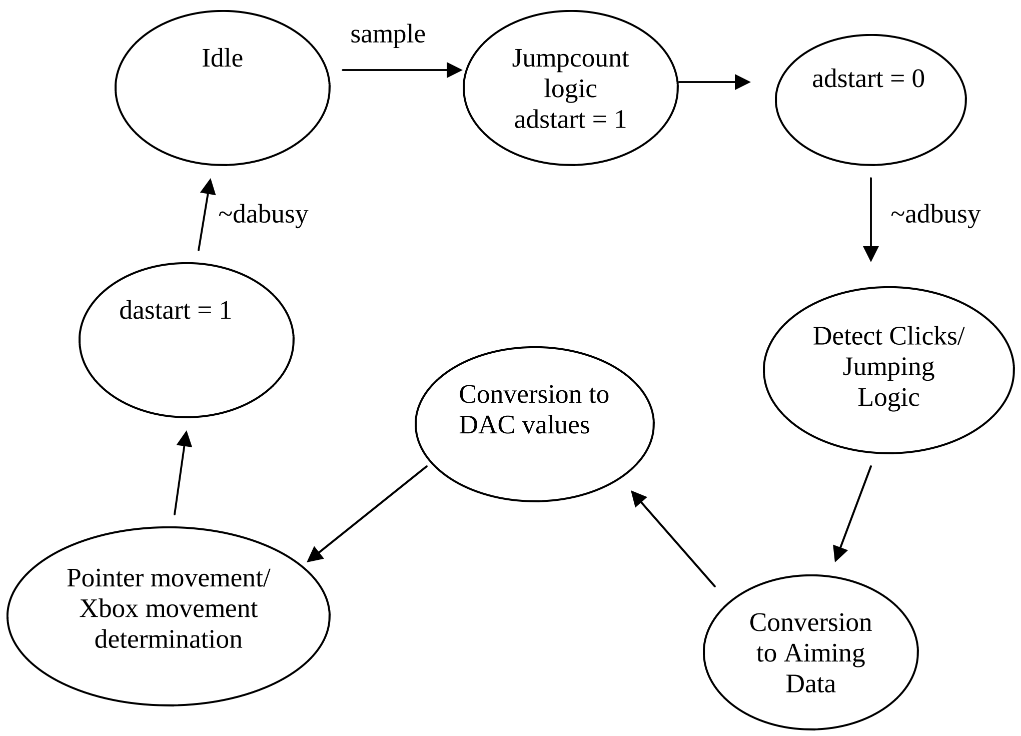


Overall Block Diagram

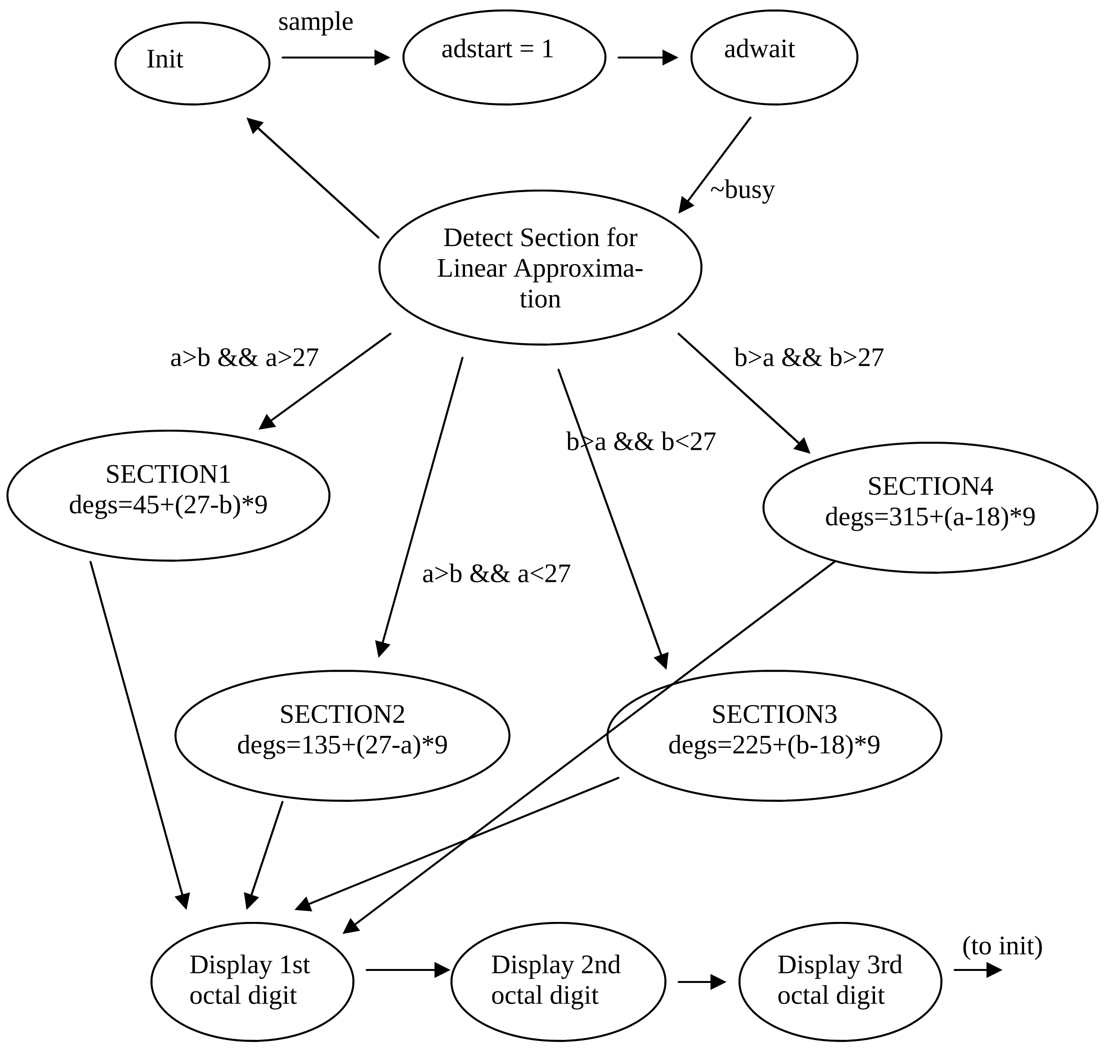



Input
Output
AD X
AD Y
XY, XZ angle lookup
Aiming Generator
16
AD Z
Detect Jump/ Crouch/Clicks
Signals for hacked Xbox controller
Signals for mouse FSM
AD compass
Rotational angle algorithm
LED matrix

Bigmajor FSM

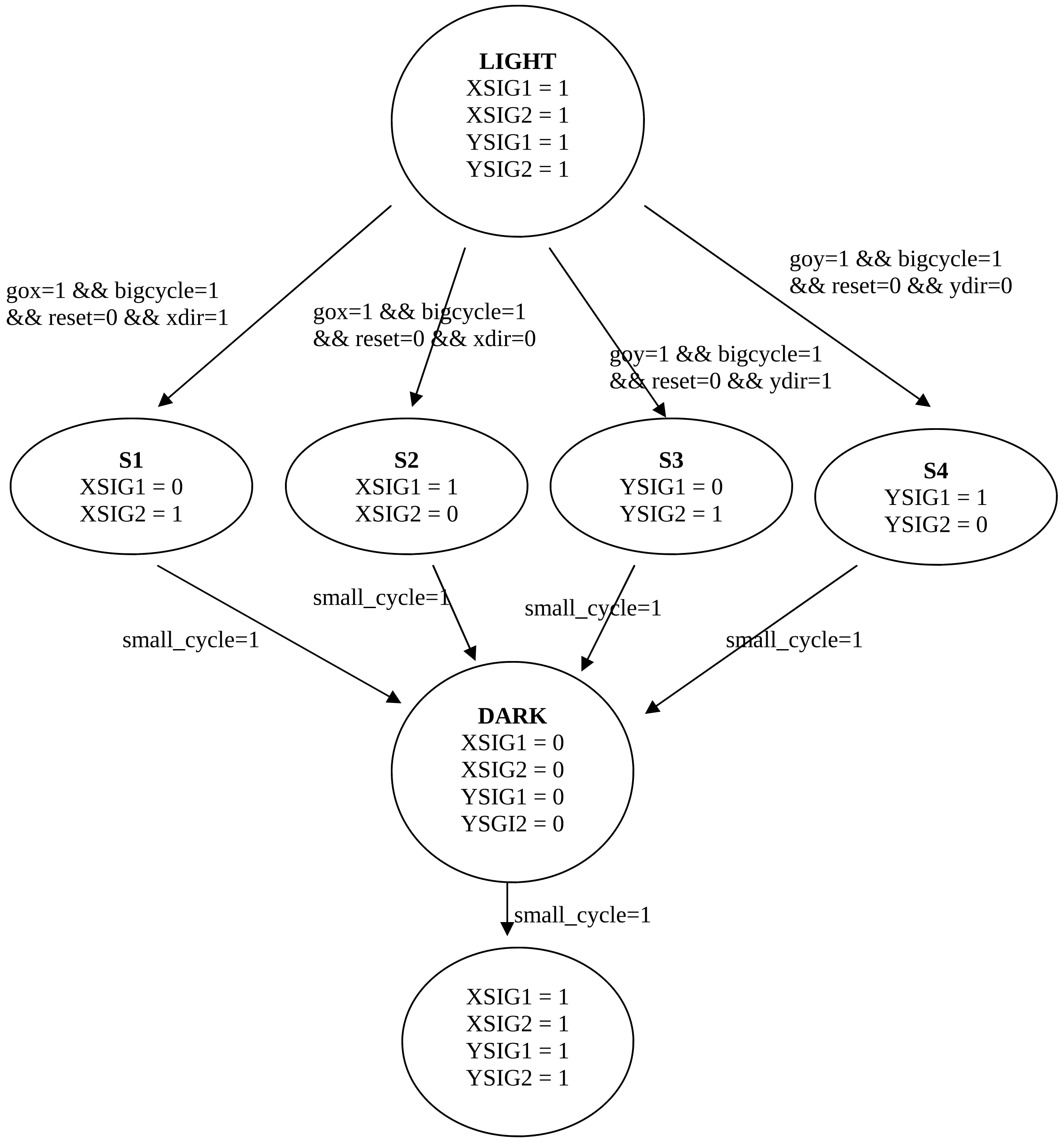

Idle
sample
Jumpcount logic adstart = 1
adstart = 0
~adbusy
Detect Clicks/ Jumping Logic
Conversion to Aiming Data
Conversion to DAC values
Pointer movement/ Xbox movement determination
dastart = 1
~dabusy

Compass FSM



sample
Init
adstart = 1
adwait
~busy
Detect Section for Linear Approxima-tion
a>b && a>27
b>a && b>27
b>a && b<27
a>b && a<27
SECTION1 degs=45+(27-b)*9
SECTION4 degs=315+(a-18)*9
SECTION2 degs=135+(27-a)*9
SECTION3 degs=225+(b-18)*9
Display 1st octal digit
Display 2nd octal digit
Display 3rd octal digit
(to init)

Optical Sensor Emulation FSM*



*This is condensed version of 2 FSMs. It is possible to be S1 or S2 AND S3 or S4 concurrently.