# PROJECT PROPOSAL

6.111 lab project

# ACTION TRACKING SYSTEM

Shubhang & Raghavendra

## 1. Overview

The Action Tracking System (ATS) is a project using FPGA to achieve real time video tracking with position feedbacks from an NTSC camera. Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing. Video tracking can be a time consuming process due to the amount of data that is contained in video. Adding further to the complexity is the possible need to use object recognition techniques for tracking.

The objective of video tracking is to associate target objects in consecutive video frames. The association can be especially difficult when the objects are moving fast relative to the frame rate. Another situation that increases the complexity of the problem is when the tracked object changes orientation over time. For these situations video tracking systems usually employ a motion model which describes how the image of the target might change for different possible motions of the object.

The system uses one camera to obtain a straight view of the field, where the target human will be. Using the 2-D

information from the camera, an object recognition module will perform threshold filtering to track the positions of various joints of the human in the view of the camera. The joints being tracked are used to reproduce a Stick figure of the human on the screen. The Action Tracking system, also involves the usage of artificial intelligence to enumerate the right connections between the joints being tracked. More information on the algorithm is given in the points_decoder module.

## 2. Background

The idea of tracking motion on a computer using a video camera has been around a couple of decades, and still is not fully perfect, because the construction of vision is a complex subject. We don't just "see"; we construct colors, edges, objects, depth, and other aspects of vision from the light that reaches our retinas. If we want to program a computer to see in the same way, it has to have subroutines that define the characteristics of vision and allow it to distinguish those characteristics in the array of pixels that comes from a camera.

There are a number of toolkits available for getting data from a camera and manipulating it. They vary from very high-level simple graphical tools to low-level tools that allow us to manipulate the pixels directly. Regardless of our application, the first step is always the same: we get the pixels from the camera in an array of numbers, one frame at a time, and do things with the array. Typically, our array is a list of numbers, including the location, and the relative levels or red, green, and blue light at that location.

There are a few popular applications that people tend to develop when they attach a camera to a computer:

**Video manipulation** takes the image from the camera, changes it somehow, and re-presents it to the viewer in changed form. In this case, the computer doesn't need to be able to interpret objects in the image, because we are basically just applying filters.

**Tracking** looks for a blob of pixels that's unique, perhaps the brightest blob, or the reddest blob, and tracks its location over a series of frames. Tracking can be complicated, because the brightest blob from one frame to another might not be produced by the same object.

**Object recognition** looks for a blob that matches a particular pattern, like a face, identifies that blob as an object, and keeps track of its location over time. Object recognition is the hardest of all three applications, because it involves both tracking and pattern recognition. If the object rotates, or if its colors shift because of a lighting change, or it gets smaller as it moves away from the camera, the computer has to be programmed to compensate. If it's not, it may fail to "see" the object, even though it's still there.

**We will be implementing object (our color markers) recognition in our system.**

## 3. Technical Approach

### Video Processing

At the most basic level, we will be obtaining a pixel's position, and its color .From those facts, other information can be determined:

- The brightest pixel can be determined by seeing which pixel has the highest color values;
- A "blob" of color can be determined by choosing a starting color, setting a range of variation, and checking the neighboring pixels of a selected pixel to see if they are in the range of variation.
- Areas of change can be determined by comparing one frame of video with a previous frame, and seeing which pixels have the most significantly different color values.
- Areas of pattern can be followed by selecting an area to track, and continuing to search for areas that match the pattern of pixels selected. Again, a range of variation can be set to allow for "fuzziness".

These approaches will be followed to process the video data and calculate the center of masses of the color markers placed on the Human.

A Pictorial representation of what the project will be doing when it is complete.

**The picture above does not show any color markers on the human, but we will be using color markers for simplicity of detection.**

## Image Reproduction

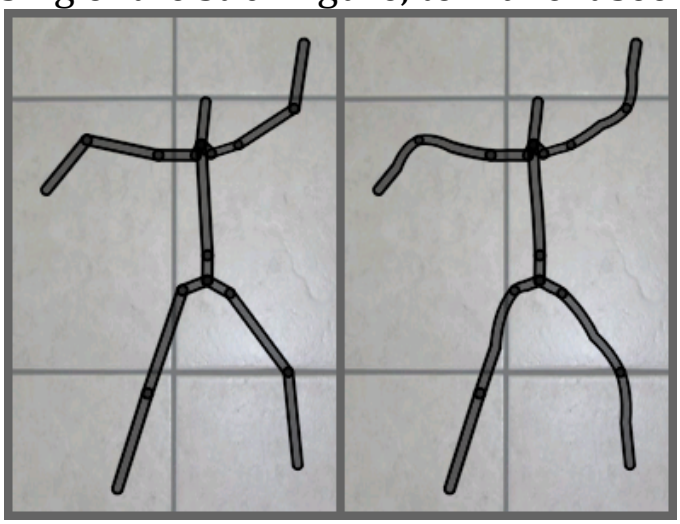
This part of the system is pretty simple when compared to the video processing part. To make the whole system interesting, we plan to make use of six markers of the same color.

This poses a big challenge and we solve it by using artificial intelligence. The system knows the various properties of the processed video output and so it makes use of these properties to figure out the right dots to connect to successfully reproduce the image of the human in the form of a stick figure.

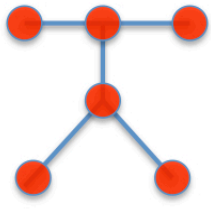
The properties are encoded into the system in the form of a module by taking into account various constraints. The system assumes that-

- Human cannot cross his/her hands or legs
- Human cannot turn around as this a 2D motion tracking.
- Human cannot put his/her hands below his/her legs.
- Human cannot join their hands/legs.

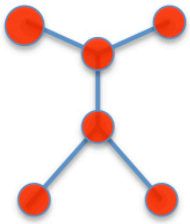
The reproduction of the image of the human is done in the form of a stick figure and more details on the algorithm are given in the module. The image reproduction also implements antialiasing of the stick figure, to make it seem more lively.



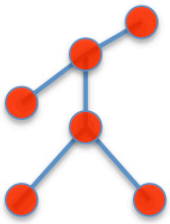
## FEW POSITIONS THAT WOULD BE TRACKED IN REAL-TIME



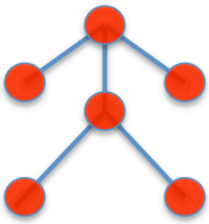
POSITION 1



POSITION 2

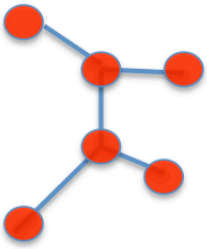


POSITION 3

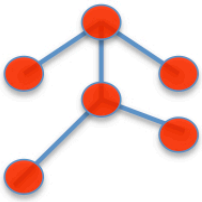


POSITION 4

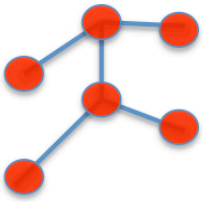
POSITION 5



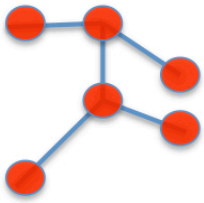
POSITION 6



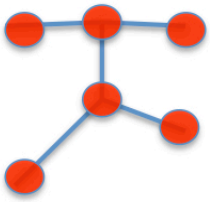
POSITION 7



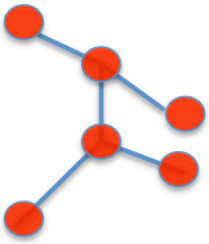
POSITION 8



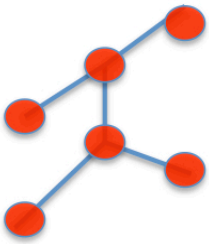
POSITION 9



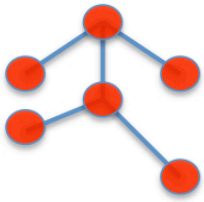
POSITION 10



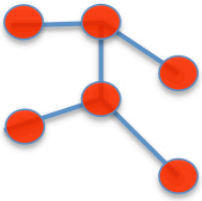
POSITION 11



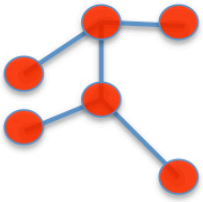
POSITION 12



POSITION 13



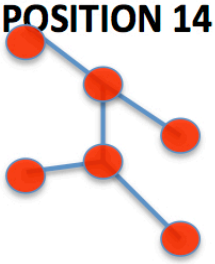
POSITION 14



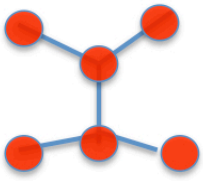
POSITION 15

POSITION 16

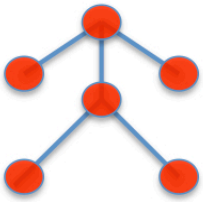
POSITION 17



POSITION 18



POSITION 19



POSITION 20

## 4. Module Information

Shubhang's Part

1. video_decoder: This module takes signals from a NTSC camera and decodes that information into YCrCb that is 30 bits long.

2. ntsc_to_ZBT: stores pixel information as RGB into the on-board ZBT memory. This module also has the RGB to HSV converter.

3. threshold_filter: filters the incoming pixels with color thresholds in order to find the position of the human joints.

Raghavendra's Part

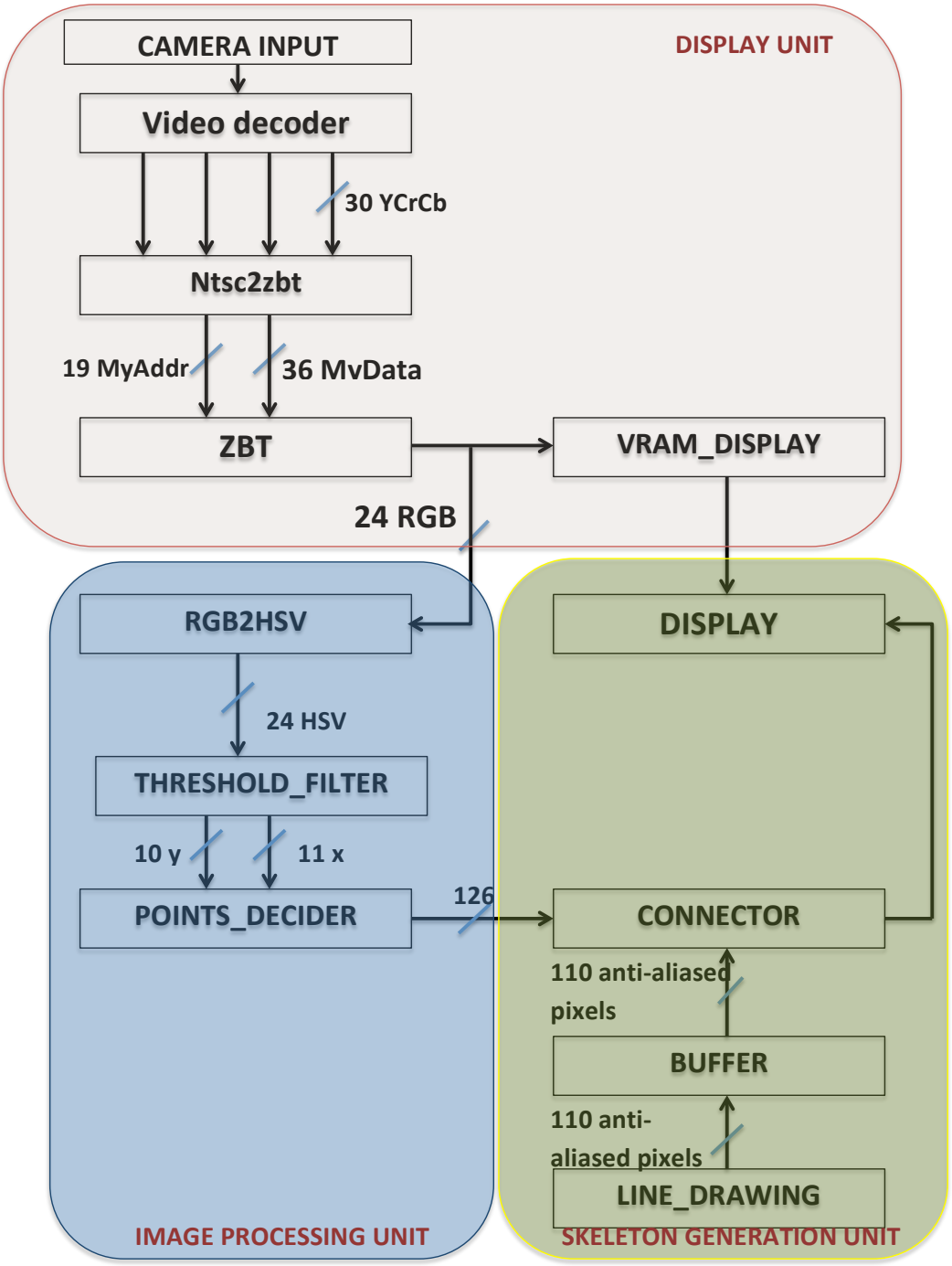
4. line_drawer : Takes 2 points in 2-D space as the inputs and connects them with a line of desired colour. It also implements anti aliasing of the line.

5. Points_Decider: With the position information of the human joints and artificial intelligence, the dots_connector module decides the respective arm , leg and torso points.

6. Connector: This module takes the arm ,leg and torso information from the points_decider module and instantiates the line_drawer module multiple times to make a meaningful skeleton.

7. Buffer: This module store the line points on the ZBT memory and makes sure that the pixels in the output do not flicker.

5. **System Diagram**



## 6. Detailed Explanation of Modules

**1. Video_decoder**: This module's job is to convert the input signals from the camera and output a 30-bit YCrCb value along with other signals such as field, vsync and hsync.

This module

1) Interfaces with the NTSC camera.

2) Decodes the input signal and converts the pixel information into a 30 bit value, with 10 bits each for Y (luma), Cr (red difference chroma) and Cb (blue difference chroma).

3) Since NTSC video standard implements interlacing mechanism, this module also outputs 'f' as a one bit value to indicate whether it is in odd or even field.

4) One bit values, 'v' and 'h' are outputted to show the position of that pixel.

5) It outputs a data_valid bit to tell other modules that the data is ready for use.

**2. ntsc_to_zbt:** This module is responsible for generating corresponding address in ZBT memory to store the current filtered incoming pixel information. Since the camera uses a different clock frequency, this module is the entry point to synchronize data from the camera with the rest of the system. A sample module is already written, but changes have to be made in order for it to store colored pixels. It accomodates a ycrcb2rgb this module converts 30bit YCrCb pixel Input into A 24bit RGB String. Such module is already written, but we note that this module has a 5 clock cycle delay between inputs and Outputs. Therefore, the address into the ZBT memory should

be delayed appropriately in order to store in the correct location.

**3. Threshold_Filter:** This is the heart of the system. The objective of this module is to do the most difficult part of the system, that is, image processing. This module processes every frame it receives from the ntsc_zbt module and figures out the existence of a reference image, in the frame. The reference image is the image of a color marker. The outputs of this module are the center of masses of various color markers. Making this module is going to be a big challenge as all the color markers are of the same color. This module uses most of the computation and memory available. It also has the highest latency.

**4. Line_drawer:** The line_decoder module implements Jack Bresenham's line drawing algorithm, described at http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm This algorithm draws a straight line between any two points on the screen, it is a particularly good choice because its implementation does not require any division. This module also implements antialiasing of a line, to make the line look more natural. The module's inputs are the two endpoints of the line segment it will draw, as well as a ready signal from the Connector module. Its outputs are the pixels to be written to the offscreen buffer, a write_enable signal to the buffer, and a ready signal back to the connector module.

The timing of the ready signals is as follows. When the line_drawer module is not in the process of drawing a segment, it asserts its ready signal. When the connector module sees this, it proceeds to iterate through entities and segments until it finds one that is not ignored. Once it finds one, it sends out the segment's data along with a one-cycle long ready signal.

Upon receiving this, the line_drawer module takes down its ready signal until it finishes processing the segment.

**5. Points_Decider:** This is the brain of the system. It receives 6 points from the threshold_filter module and figures out the right points to be joined to make a meaningful skeleton on the screen. This module assumes a few constraints- the human cannot cross hands, cross legs, put hi/her hands below his/her legs and finally cannot turn around. This modules functions using artificial intelligence, it uses the fact that the torso of the human is always situated in the center of the left and right side of the body. It has a latency of 8 clock cycles and does not have any start or stop signal as the buffer in the threshold_filter module makes sure that all the points are synchronized. Making this module is going to be a fair amount of challenge as there are a lot of possible combinations of orientations of hands and legs.

**6. Connetor:** This module receives the coordinates of hands, legs and the torso from points_decider module and joins the legs and hands to the right points of the torso, by simply instantiating the line_drawer module multiple times. This module basically is not affected by any timing, it always gives the right output as the inputs are constant.

**7.Buffer:** The buffer module allows random-access pixels to be displayed to the screen without needing huge arrays of combinational logic and without causing flicker due to erased pixels. It physically interfaces with the Virtex2's two ZBT memories, clearing and then writing to one while the other is displayed on-screen. This module is synchronized with the SVGA module's hcount, vcount, and vsync signals and operates on a two-frame period, bounded by vsync. During the first frame of the period, hcount and vcount are used as indices to

the off- screen buffer and the value at each address is set to zero. During the second frame, the module accepts pixel coordinates and color values from the line_drawer module and writes them to the off-screen buffer. During both frames, hcount and vcount also index the on-screen buffer, the contents of which are sent to the VGA interface. Upon the rising edge of the vsync signal, the off-screen buffer becomes on-screen and vice versa.

## MILESTONES

- Till date, everything thing other than marker detection has been completed and tested.
- 11/15 Figure out the best color for Detection
- 11/16 Figure out the range of H,S,V for the color to be detected.
- 11/17,18 Make code for marker detection.
- 11/21 Test Code and make changes if required.
- 11/23 Put all the modules together.
- Plan for Implementation of Extensions.