Vehicle Control using Video Surveillance

1. Overview

The Vehicle Control using Video Surveillance (VCVS) is a project using FPGA to achieve real time vehicle control with position feedbacks from camera. The system uses one camera to obtain an eagle-eye view of the field, where the target vehicle will be. Using the 2-D information from the camera, an object recognition module will perform threshold filtering to find the position of the vehicle and any obstacles on the map. VCVS provides a user interface that allows users to choose the desired destination of the vehicle and the central control FSM will direct and vehicle to reach the final destination with the presence of obstacles in the way. The VCVS system includes the following main modules.

Jorge'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.

3. **user_input**: mouse control for user interface, and user interface features

4. **grid_map**: generates a grid map that virtualizes the field, displays the current position of the vehicle, updates the locations of obstacles, and shows final

Kevin's part

5. **YCrCb_to_RGB**: converts 30 bit YCrCb to 24 bit RGB.

6. **threshold_filter**: filters the incoming pixels with color thresholds in order to find the position and orientation of the vehicle. The threshold filter also filters out the obstacles and stores that information in a BRAM.

Krishna's part:

7. **main_control_FSM**: with the position and orientation information of the vehicle, desired destination from user input, and access to BRAM of obstacle information, the main control FSM decides how to control and vehicle, and the optimal path it should take to reach the destination. The control FSM will send control signals to the interface that controls the vehicle.

2. Description of Modules

This section explains the functions of each module in more detail, including the descriptions of inputs and outputs, test strategy, and potential challenges.

1. **video_decoder**: This module interfaces with the NTSC camera, decodes the input signal and converts the pixel information into a 30 bit value, with 10 bits each for Y, Cr and Cb. 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. One bit values, v and h are outputted to show the position of that pixel. 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 slight changes have to be made in order for it to store colored pixels.

3. **user_input**: This module observes the current coordinates of the mouse pointer and the status of the mouse buttons. The module will output signals corresponding to desired destination that user selected. We will use a ps/2 mouse.

4. **grid_map**: This module creates a grid map representing the field camera is looking at. According to information such as vehicle position, obstacle position, and user desired destination, grid map highlights them with different colors. Coordination and orientation of vehicle position will also be updated continuously as vehicle moves in the field. Depending on how good we want our interface is visually, a lot more features could be added later on. For a start, we will do minimal visual design and use it as mainly a debugging tool.

5. **YCrCb_to_RGB:** 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.

6. **threshold_filter**: This module reads directly from the incoming video data; an YCrCb to RGB converter is embedded in this module in order to store RGB information in memory. Then this module filters each pixel using a simple color threshold. On the vehicle there will be markers for both the front and rear, we find the center of mass of each marker so that we can find the position as well as the orientation of the vehicle. For obstacles, we do the same filtering except we store that information in a BRAM. The main FSM will ask if an obstacle is present in a location by passing in the location as the address in the BRAM, and a one bit value will be outputted. This mechanism might be slow and finding obstacles in the field is also extremely challenging since we can’t do it on a pixel by pixel

basis due to noise from camera. Besides, if we are assuming dynamic environment, i.e. obstacles can also change their locations at real time, then clearing the BRAM at each new frame might also take some time, and possibly we won't have enough time to clear the memory.

7. **main_control_FSM**: This is the brain of the system, which takes in position and orientation of the vehicle, desired location from the user, positions of obstacles, and directs the vehicle to reach the desired destination. The challenge is first the physical limitations in the vehicle itself and communication with the vehicle. We plan to purchase a remote control car that is able to locally rotate 360 degrees, and we are essentially interfacing with the remote control to ask it to send appropriate signals for us. This interface might take some time to understand and rigorous experiments and tests will be carried out to make sure that the vehicle will be controlled correctly. Computation power could be another limiting factor in this project; as a result, we restrict our cars to only move in 90 degree directions, therefore simple algorithms will be implemented first. For future improvements, more search algorithms could be implemented to achieve real optimal path.

3. VCVS Block Diagram