Autonomous Tracking and Intercepting Vehicle Utilizing Image Processing

Project Team:
Mike Barngrover
Peter Knaub
Andrew Lovitt

Advisors:
Dr. Brian Huggins
Dr. Aleksander Malinoski

May 16, 2002
Abstract

This project was the implementation of an autonomous vehicle, which tracks and intercepts a moving target of predetermined color. The system consists of a camera, a PC, controller, and electro-mechanical sub-systems. The method of identifying and tracking the target is image processing (IP). The IP algorithm on the PC uses imagery input from the camera to locate the center of gravity of the color and sends a signal to the micro-controller to move the camera so as to keep the target centered in the camera's field of view. After the target has been identified, the vehicle will move to intercept it. The vehicle control algorithm, which runs on the micro-controller, adjusts vehicle direction so as to continuously decrease the angle between the camera bore sight and the vehicle bore sight. Vehicle speed is kept constant. Implementation of the system and results are discussed.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Functional Description</td>
<td>1</td>
</tr>
<tr>
<td>System Specifications</td>
<td>3</td>
</tr>
<tr>
<td>System Design</td>
<td>4</td>
</tr>
<tr>
<td>Image Processing</td>
<td>4</td>
</tr>
<tr>
<td>Camera Control</td>
<td>11</td>
</tr>
<tr>
<td>Vehicle Control</td>
<td>20</td>
</tr>
<tr>
<td>Implementation, Results, and Evaluation</td>
<td>23</td>
</tr>
<tr>
<td>Image Processing</td>
<td>23</td>
</tr>
<tr>
<td>Camera Control</td>
<td>25</td>
</tr>
<tr>
<td>Vehicle Control</td>
<td>26</td>
</tr>
<tr>
<td>Conclusions</td>
<td>27</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>27</td>
</tr>
<tr>
<td>Appendixes</td>
<td></td>
</tr>
<tr>
<td>Micro-controller Software Flowcharts</td>
<td>A</td>
</tr>
</tbody>
</table>
Introduction

The goal of this project was to build an autonomous vehicle, which tracks and intercepts a moving target with predetermined characteristics. The object’s characteristic to track was decided to be color. The method of identifying and tracking the target is image processing (IP). The IP algorithm on a personal computer (PC) uses imagery input from the camera to locate the center of gravity of the color. The PC then sends a signal to the micro-controller to move the camera so as to keep the target centered in the camera's field of view. After the target has been identified, the vehicle will move to intercept it. The vehicle control algorithm, which runs on the micro-controller, adjusts vehicle direction so as to continuously decrease the angle between the camera bore sight and the vehicle bore sight. The bore sight is defined as an imaginary line drawn in the direction the object is pointing from the object. Vehicle speed is kept constant. Implementation of the system and results are discussed.

Functional Description

The system consists of a camera, a personal computer (PC), micro-controller (µC), Field Programmable Gate Array (FPGA), electro-mechanical, and power distribution sub-systems. A system-level block diagram is shown in Figure 1.

Upon system start-up, the system goes into acquire mode. In the acquire mode the camera is repeatedly swept through its rotational range until the IP algorithm locates the target. The IP algorithm on the PC locates the target color by processing color data from the camera and comparing it to a reference color. The user predefines the reference color by entering the numerical Red, Green, and Blue values (RGB vector) into the IP program.

After the target has been identified, the system switches to bore-sight mode to track and intercept the target. The PC sends a signal to the micro-controller representing the amount and direction the camera needs to rotate in order to keep the target centered in the camera’s field of view. Once the micro-controller receives this signal from the PC, it reads the current position of the camera from the FPGA, adds the desired change in position, resulting in the desired new position of the camera, and writes that new position to the FPGA. Upon receiving the new camera position from the micro-controller, the FPGA drives a stepper motor through power electronics to move the camera to the desired position while keeping track of the camera’s current position. The vehicle control algorithm on the micro-controller attempts to minimize the angle between the camera bore sight and the vehicle bore sight by adjusting the position of the steering wheels while maintaining a constant speed on the drive wheels.

The camera sends imagery data to the PC across a USB connection via a TWAIN interface. The PC communicates with the micro-controller through a serial connection. The FPGA is connected to the micro-controller via bus expansion.
The system is powered by two 12 VDC lead-acid batteries. One battery provides power for the μC, FPGA, stepper motor, and linear actuator. The FPGA board has a 7805 DC regulator on board to provide 5 V, however, a 7809 is also used to decrease the voltage to 9 V before entering the FPGA board to minimize power dissipation in the 7805. The 7809 has a 1 μF capacitor across its output to ground to filter out noise. The μC board has a 7805 with a heat sink, so a no 7809 was used for its power. The second battery powers the drive motors.

![System-level Block Diagram](image)

The camera used in the project was a Logitech QuickCam® Pro 3000 webcam. It has a resolution of 640x480 pixels and a viewing angle of 20° on each side of center. The QuickCam® is equipped with a USB cable to connect to the PC. The PC used was a Sony Viao® notebook computer with a 1 GHz AMD Athlon4™ processor, 256 MB of DDR266 RAM, and a 20 GB hard drive. The micro-controller is a Seimen's 80535 (Intel 8051 family) running at 11.0592 MHz located on an EMAC MicroPac535 development board. A Xilinx 4000E series 4010PC84 FPGA on a XS40 development board was used for the camera controller. A uni-polar 4-phase 12 VDC 1.8°/step stepper motor with a holding torque of 2000 g·cm and a phase current of 400 mA was used to position the camera. The linear actuator used to steer the vehicle was a Warner Electrak1 with feedback. The actuator is also a 12 VDC device. The drive motors are the stock motors on the vehicle, which is a Fisher Price Power Wheels® Gaucho Jeep children's vehicle.
System Specifications

Intercept Vehicle Specifications:

The vehicle is initiated into pursuit of the target when the user enters a velocity via the keypad. When the vehicle has successfully intercepted the target, the user stops the vehicle via the keypad. The user determines when the vehicle has successfully intercepted the target.

<table>
<thead>
<tr>
<th>Min. Value</th>
<th>Max. Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Velocity:</td>
<td>5 mph</td>
</tr>
<tr>
<td>Vehicle Turning Radius:</td>
<td>10 ft</td>
</tr>
<tr>
<td>Steering Time Lock-to-Lock:</td>
<td>3 sec</td>
</tr>
<tr>
<td>Battery Life:</td>
<td>1 hour 3 hours</td>
</tr>
<tr>
<td>Camera Time from Full CCW to Full CW:</td>
<td>&lt;1 sec</td>
</tr>
<tr>
<td>Operation Time of Stepper Motor w/o Fan:</td>
<td>30 min</td>
</tr>
<tr>
<td>Operation Time of Stepper Motor w/ Fan:</td>
<td>10 hours</td>
</tr>
<tr>
<td>Image Capture Rate:</td>
<td>0.75 sec 2 sec</td>
</tr>
<tr>
<td>Distance to Target:</td>
<td>4 ft 20 ft</td>
</tr>
</tbody>
</table>

Target Specifications:

This system was designed to track and intercept a target of predetermined color. The value, a RGB vector, of the color of the target must be know to within 5% for each component of the vector. In addition, the color of the object cannot vary by more than 10% for each color vector component. The image seen by the camera must have the target stand out against the background. The more contrast between the background and the target, the better the vehicle will be able to track it. The target must move with a velocity no greater than one half of the velocity of the vehicle and not turn sharper than the turning radius of the vehicle.
System Design

The design of the system was broken into three parts: image processing, vehicle control, and camera control. The system control consists of a major loop and a minor loop. The major loop is comprised of the image processing and camera controller and the vehicle controller is the minor loop. The control block diagram for the system is shown in Figure 2. The IP determines how far the camera needs to rotate and the camera controller rotates the camera to the new position. The vehicle control attempts to align the vehicle with the camera by steering in the direction of the camera’s bore sight, thus adding the rotation of the camera. The IP compensates for this additional rotation in the next camera movement calculation. If the camera bore sight and the vehicle bore sight are aligned, the vehicle will proceed straight forward.

To conserve paper no, code listings are included with this report. The code listings are available on the accompanying CD-ROM or on the project website located at: http://cegt201.bradley.edu/~projects/proj2002/homers/.

Image Processing

The image processing consists of four major software modules. These modules comprise the major loop of the System Control Block Diagram. The program modules are: camera interface, image processing, command creation, and communication. All the modules were programmed in C++ to aid in ease of use and testing. They were also created in C++ to make it easier to update and embed in other applications. The software flow chart is seen in Figure 3.
When the program starts, the class Senior Project is called. The first operation the class performs is to prompt the user for the reference pixel in RGB vector format. The class then starts the initialization of the TWAIN interface. Once the TWAIN libraries are loaded, the camera software is loaded and the program enters the main loop. The main loop takes an image from the camera through the TWAIN interface. Once the image is acquired, the program performs a cross product of the image with the reference pixel. When this is completed the entire image is transformed along with the reference pixel. The cross product is performed on the new data and the two cross product results are added together. The image processing now creates an array of values telling how well a 0.5° section of the image correlated to the reference pixel. Once this array is completed the command signal is determined by the command class and sent to the micro controller.

**Reference Pixel**

The reference pixel is used to determine whether an object in the image captured is part of the object it is trying to detect. The pixel is in RGB format. This means there are three numbers, each 0-255, for the red, green, and blue channels. These represent how much of one color is present in that pixel. The two extremes are when red, green, and blue are 255 or 0. These represent white and black respectively.
All other colors are a different combination of red, green, and blue. The camera will automatically find the brightest intensity in the image and normalize that pixel to white. This causes problems if the image processing moves from dim light to bright light. When this happens the object will change colors because of the change in lighting. The change in lighting can cause the image processing not to function correctly.

Before the image processing can start, the user must determine what the values of the RGB vector should be for the object. This is found by taking a picture of the target with the camera software package. Then the picture needs to be opened in a program where the user can find out the value of pixels. The user needs to pick a pixel that looks to be average color of the object. This RBG vector is used as the reference pixel.

Camera interface

The software to run the camera interface is an implementation of a TWAIN interface. The cameras acquisition software is run in the background on the laptop. This keeps the camera on and in constant acquire mode. The software then makes calls to the TWAIN libraries to retrieve the image. The TWAIN interface takes care off all the calls to the camera and allocating the space for the image in memory. The TWAIN interface class also keeps track of all variables passed between the TWAIN DLL and the software. The software is arranged so the states are cycled through in the correct order as defined by the TWAIN specification. If these are call in the wrong order the class will generate an error. There are also specific errors for when the camera is not found and low memory errors.

Image Processing

The image-processing module takes care of all of the graphic manipulation. The image is passed from the TWAIN interface back to the main module. The main module then passes the reference to the image and the image processing module starts. The first process the algorithm completes is a cross product of the reference pixel and each pixel in the image. The cross product is defined as:

Reference pixel: \(<r_1 \ g_1 \ b_1>\)
Picture pixel: \(<r_2 \ g_2 \ b_2>\)
Cross Product: \((g_1*b_3-b_2*g_2)-(r_1*b_3-r_3*b_1)+(r_1*g_2-r_2*g_1)\)

The code segment is seen below:

```
// performs the crossproduct
//\(x\), \(y\), and \(z\) are reference red, green, and blue respectably
long IP::vectorproduct(int y1, int y2, int y3)
{
    int x,y,z;
    x=staticx*y3-y2*staticz;
    y=-(staticx*y3-y1*staticz);
    z=staticx*y2-y1*staticy;
    if(x<0) x*=-1;
```

if(y<0) y*=-1;
if(z<0) z*=-1;
return x+y+z;
}

The value of the cross product using this algorithm is the amount of linear dependence between the vector starting at the origin through the reference pixel and the vector string at the origin through the picture pixel. A small number represents more correlation with a value of 0 being exactly linearly dependent. Linear dependence is important because it tells how close to the reference pixel the picture pixel is.

The result of the first cross product is that it will provide a set of pixels, which their vectors are linearly dependent to the reference pixel. Each of these pixels has a correlation value of zero. Figure 4 displays the results of the first cross product. The black line represents all points with a correlation value of zero.

![Figure 4](image)

Figure 4. This is a 2-D representation with only the red and green components. All points along the black line has a cross product of zero with the reference pixel.

Next every pixel in the image and the reference pixel are transformed using the following equation for each component of the picture. Each of these new values is written over the old ones in memory because the original values are no longer needed.

```
New value = Old Value + 0.5*(255-Old Value)
```
The code the implements this is:

```c
static x = static x + .5 * (255 - static x);
static y = static y + .5 * (255 - static y);
static z = static z + .5 * (255 - static z);
// change the picture
for (int i = 0; i < 640 * 480 * 3; i++)
{
    picture[i] += .5 * (255 - picture[i]);
}
```

The consequence of this is a non-linear transform. The reason this is non-linear is because if we look at the vector through the original point and the origin, this will cause the vector to move off of this line by a different amount depending on how close the pixel is to the origin. This is seen visually in Figure 5. The points on the first linear dependence line get transformed to different places. As the original points get further away from the reference point, their transform gets further away from the transformed linear dependence line. This is desirable because the further from the original reference pixel a point is the less it needs to be correlated with the transformed dependence line. So the final image processing will create an elongated square in the vector space. This is seen in Figure 6.

Figure 5. Shows the mapping of the points on the first linear dependence line to the new positions after the transform has been applied.
Figure 6. This shows the correlation levels in a 2-D vector space. The three levels of correlation are seen around the reference pixel. The dashed line is a brightness line.

An advantage of this image processing is also seen in Figure 6. The dashed line in Figure 6 represents a brightness line. As the brightness of the object increases, the color values increase non-linearly. The cross product algorithm for the image processing maximizes the detection across the brightness line with little other pixels being correlated as well. This allows the image processing to detect more change in the color of the object.

Once the cross product is taken a second time, both cross product values are added together. Now the image is broken up into 80 columns, each represents 0.5 degrees of viewing. Each of the 80 columns of the image is matched with a 5-value array. This array will keep track of how many pixels are correlated at different levels with the reference pixel. Each pixel value in the 0.5 degree slice of the image is representative of one of the 5 ranges. When a pixel falls into one of the array values, that array value is incremented by one. A graphical view of the array for each column is seen in Figure 7. After the entire picture is evaluated in this way, an 80 by 5 array is passed back to the main module.
Command Creation

The command creation module uses the eighty-by-five matrix made in the image processing and turns it into a command signal. The module creates a weighted value for each column of the array. The weighted value is computed by looking at the column and the columns up to two away on each side, if these are over the bounds of the array the value of that column component is zero. The weight is computed as follows:

\[
\text{Weighted value} = (\text{Place 1 this column and 2 columns on each side}) + 0.25\times(\text{Place 2 of this column and one column on each side}) + 0.1\times(\text{Place 3 of this column})
\]

The module then locates the column with the highest weighted value. This corresponds to the angle of the center of the image. Each matrix column is 0.5 degrees. The camera views 20 degrees on each side so the 40th column is the dead center of the camera. If the perceived value is less than about 2.5 degrees on either side of the center, no command is sent. This prevents the feedback control from oscillating.

The module of the image processing will detect the left side (from the camera’s view) of the object. This module takes into account that the object will fill about 2 degrees of the camera view. The column number minus 39 is returned from this module. If there is no column with any correlation to the object to detect the command module will send a command of –39, which is the left edge of the image.
Communication

The communication module takes the command from the command creation module and places it on the serial port. This value is then turned into the number of stepper motor steps and sent to the micro controller. The command is constructed so the minimal amount of computations is done on the micro-controller. The information sent to the micro-controller is the number of steps, defined by the stepping motor, to turn. If the number is negative the camera will move left, and if the number if positive the camera will move right. The communication is done over a serial link directly to the micro-controller.

Another operation this module performs is to make sure the camera will not oscillate. The module does this by limiting the camera movement to ten degrees in either direction. The module will also not send a command that is less than two degrees on either side to prevent oscillation.

Implementation

The first major problem encountered was the TWAIN interface. The TWAIN drivers written by the companies that develop the web cams do no correctly implement the TWAIN specification. This caused the software that interfaces with the TWAIN specification to take a long time to create. The main problem with the TWAIN drivers was that they would not support a disabled GUI. This was a major problem because the car is automated and if the GUI came up a user would need to click every time a picture needed to be taken.

The image processing was developed in two iterations. The first iteration was developed early on. It was apparent early that this would not work. The image processing just took the average of the green and blue channels and subtracted it from the red. This caused a lot of problems because the color that was being detected was never close to pure red. In addition, large objects of other colors would distort the image processing.

The second iteration developed is the one described in this paper. This was a lot more robust and could take any pixel as a reference pixel to correlate to. This allowed the image processing to be tested with different shades of red in addition to different colors.

Camera Control

The camera control system resides in both the micro-controller and the FPGA. The micro-controller handles the communication between the PC and the FPGA. The micro-controller also sends reset and calibrate signal to the FPGA upon start-up. The FPGA takes the position command signal from the micro-controller and converts it to drive signals for the stepper motor while keeping track of the current position of the camera.
The FPGA controller is broken into three sub-systems: bus logic, controller, and sequencer. Figure 8 shows the block diagram for the FPGA sub-system. Each sub-system was implemented in a VHDL entity. Once each sub-system was designed and tested separately, they were instantiated into a top-level entity called Stepper_Controller.

The bus logic entity is represented by the “Address Decoding And Bus Drivers” block. The bus logic entity has eight bi-directions bits, which are connected to the \( \mu \)C bus, eight input bits for reading the current camera position from the controller, eight output bits to write the desired position to the controller, and address lines: A8, EXTIO’, RD’, and WR’. When the \( \mu \)C writes to the FPGA, A8 \( \rightarrow \) Hi, EXTIO’ and WR’ \( \rightarrow \) Lo and the bus logic latches in the 8-bit data from the bus as the “GOTO” value for the controller. When the \( \mu \)C reads the current from the FPGA, A8 \( \rightarrow \) Hi, EXTIO’ and RD’ \( \rightarrow \) Lo and the bus logic puts the 8-bit data from the AT register in the controller out on the bus.

The controller entity’s main role is to keep track of the current position, “AT”, and the desired position, “GOTO”. From the current and desired positions, the controller determines how to move the motor. If GOTO is greater than the AT, the motor is to be moved clockwise. If the GOTO is less than the AT, the motor is to be turned counterclockwise. The controller has three outputs that go to the sequencer: direction, enable, and clock. The direction bit indicates which direction to move the motor, “1” indicates clockwise, “0” indicates counterclockwise. Since the direction bit always indicates a direction, the enable bit is cleared when the AT and GOTO are equal. The
The clock signal is used by the sequencer to create its output sequence. The controller divides down an input clock of 1 MHz to one of three frequencies depending on the mode the controller is in: 250 Hz if the amount to be moved is greater than 5 steps, 125 Hz is the amount to be moved is 5 steps or less, and 14 Hz for calibration. The 1 MHz input clock signal is generated by a external crystal oscillator.

The AT register value must to be initially calibrated upon start-up to have a reference point from which to start counting. The 0 position of the camera bore sight is approximately 5° in the CW direction from straight back on the vehicle. The µC sends the FPGA the calibrate signal, which is described in the micro-controller design section. When the calibrate input is high, the controller sets direction to “1” and divides down the clock to 14 Hz, so as to slowly turn the motor clockwise. An optical switch positioned, so that, when the camera’s position is 351°, the optical switch is triggered the controller stops the motor and loads “11000011” into AT. 11000011 is 195 decimal, which is 351°/1.8° per step, so there are 196 steps from 0° position to 351° position. Figure 9 shows the diagram for the calibrate mechanism. A Fairchild H21A1 is used for the optical switch. A transparent disc with a 5° opaque wedge is placed under the camera on the motor shaft. The transparent portion allows photons projected from the transmitter to hit the detector pulling the limit signal low. When the opaque wedge blocks the photons, the detector transistor cuts-out and the pull-up resistor pulls the limit signal high. Figure 10 shows the optical switch circuit diagram. The value for resistor R1 was picked, such that, the transmitter current is approximately 30 mA. R2 was pick so that the maximum sink current of the detector is 1.8 mA.

Figure 9. Diagram for Calibration Mechanism
The sequencer entity receives the direction, enable, and clock signal from the controller and generates the proper drive signals for the stepper motor. The sequencer is a simple logic design derived from the truth table for the stepper motor shown in Table 1. Figure 11 shows the logic circuit diagram for the sequencer. This diagram was then converted into VHDL code to be placed on the FPGA.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Truth Table for Stepper Motor Drive Signals
Figure 11. Logic Circuit Diagram for Sequencer

The final signal and pin assignments are contained in Table 2.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Pin #</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A8_in</td>
<td>P5</td>
<td>INPUT</td>
<td>Expansion bus address line A8</td>
</tr>
<tr>
<td>Cal_in</td>
<td>P4</td>
<td>INPUT</td>
<td>Calibrate mode input</td>
</tr>
<tr>
<td>Clk_in</td>
<td>P35</td>
<td>INPUT</td>
<td>External clock input</td>
</tr>
<tr>
<td>Data_bus&lt;0&gt;</td>
<td>P18</td>
<td>BIDIR</td>
<td>Expansion bus data line A0</td>
</tr>
<tr>
<td>Data_bus&lt;1&gt;</td>
<td>P19</td>
<td>BIDIR</td>
<td>Expansion bus data line A1</td>
</tr>
<tr>
<td>Data_bus&lt;2&gt;</td>
<td>P20</td>
<td>BIDIR</td>
<td>Expansion bus data line A2</td>
</tr>
<tr>
<td>Data_bus&lt;3&gt;</td>
<td>P23</td>
<td>BIDIR</td>
<td>Expansion bus data line A3</td>
</tr>
<tr>
<td>Data_bus&lt;4&gt;</td>
<td>P4</td>
<td>BIDIR</td>
<td>Expansion bus data line A4</td>
</tr>
<tr>
<td>Data_bus&lt;5&gt;</td>
<td>P25</td>
<td>BIDIR</td>
<td>Expansion bus data line A5</td>
</tr>
<tr>
<td>Data_bus&lt;6&gt;</td>
<td>P26</td>
<td>BIDIR</td>
<td>Expansion bus data line A6</td>
</tr>
<tr>
<td>Data_bus&lt;7&gt;</td>
<td>P27</td>
<td>BIDIR</td>
<td>Expansion bus data line A7</td>
</tr>
<tr>
<td>Extio_n</td>
<td>P38</td>
<td>INPUT</td>
<td>Expansion bus address line EXTIO'</td>
</tr>
<tr>
<td>Lim_in</td>
<td>P28</td>
<td>INPUT</td>
<td>Calibration limit signal</td>
</tr>
<tr>
<td>Memory</td>
<td>P61</td>
<td>OUTPUT</td>
<td>XS40 expansion board SDRAM output enable line</td>
</tr>
<tr>
<td>Micro</td>
<td>P36</td>
<td>OUTPUT</td>
<td>XS40 expansion board micro-controller reset line</td>
</tr>
<tr>
<td>Rd_n_in</td>
<td>P39</td>
<td>INPUT</td>
<td>Expansion bus address line RD’</td>
</tr>
<tr>
<td>Rst_n_in</td>
<td>P3</td>
<td>INPUT</td>
<td>FPGA reset’ line</td>
</tr>
<tr>
<td>Wr_n_in</td>
<td>P40</td>
<td>INPUT</td>
<td>Expansion bus address line WR’</td>
</tr>
<tr>
<td>X_out</td>
<td>P9</td>
<td>OUTPUT</td>
<td>Stepper motor drive signal X</td>
</tr>
<tr>
<td>Xbarout</td>
<td>P8</td>
<td>OUTPUT</td>
<td>Stepper motor drive signal X’</td>
</tr>
<tr>
<td>Y_out</td>
<td>P7</td>
<td>OUTPUT</td>
<td>Stepper motor drive signal Y</td>
</tr>
<tr>
<td>ybarout</td>
<td>P6</td>
<td>OUTPUT</td>
<td>Stepper motor drive signal Y’</td>
</tr>
</tbody>
</table>

Table 2. Pin Assignment and Descriptions for FPGA
The micro and memory pins are for the XS40 board specifically. Pin 36 of the FPGA is connected to the reset input of the on-board µC and pin 61 is connected to the output enable of the on-board SDRAM. By assigning these pins a high level, the µC is put into reset and the SDRAM’s outputs are disabled. This is not necessary for the current pin configuration, but it was done as an experiment for a professor.

**Power Circuitry**

The FPGA is connected to the stepper motor via power electronics as shown in Figure 12. The transistors saturate and cut-off to switch the current flow through the motor windings. When the transistor is on, the winding is energized. Each motor winding has a current draw of 400 mA. 2N2219A’s are used for the transistors for their 0.8 A maximum current rating.

The purpose of R1 is to bias the transistors so as to ensure that the transistors only operate in either cut-off or saturation. The value of R1 was chosen to be 1.1 kΩ. This allows for a base current of 10.3 mA as given by the following equation:

\[ i_b = \frac{(12 - 0.7)}{1100} \]

The 2N2219A has a minimum hfe value of 100. This puts the collector current, theoretically, over 1 A, which is more than double the current draw of the motor.

The diodes between the collector and the +12 v rail are free-wheeling diodes. The diodes suppress the voltage spikes, which occur when switching an inductor, allowing current to flow back to the +12 v supply when the collector voltage exceeds 12 v. This prevents the collector-emitter voltage from exceeding the breakdown voltage of the transistor, which is 20 v for the 2N2219A. 1N270 Germanium diodes are used because of their low \( v_d \) characteristic of 0.2 v.

There is also a snubber circuit, C and R2, below the diode. The diode cannot react quickly enough to dissipate the over voltage, so the snubber circuit absorbs the initial voltage spike until the diode can turn on and dissipate it. The values of C and R2 where picked to be 10 \( \mu \)F and 47 Ω respectively.

The 7407 IC is an open collector driver. The purpose of the 7407 is to divert current away from the transistor to turn it off. When its input is logic “1”, the output transistor of the 7407 cuts off, allowing base current to the transistor. When its input is logic “0”, the output transistor of the 7407 saturates, diverting current from the base of the transistor. The 7407 has a maximum sinking current rating of 0.25 A, which is more than twice the actual amount of current sanked.

The power switch in the circuit is used to turn off the stepper motor circuitry without having to cut out the power to the FPGA board. The colors indicated by the wires connected to the motor windings are the colors of the leads on the motor.
Micro-controller Design

The software that runs on the μC is broken into three modules: startup, keypad, and communications. Upon start-up, the μC immediately resets the FPGA. Once the FPGA is in reset, the initialization software is run to set-up the proper register settings for serial communication, bus expansion, and interrupts. COM1 is used for the serial communications with the PC. The startup module calls the initialization routine for the serial port, which resides in the communication module to maintain mobility. External Interrupt 3 is used to detect the limit signal and is set to rising-edge triggered. More detail on the function of these signals is discussed in the “FPGA Design” portion of this report. P4.4 and P4.5 are used to send the reset and calibrate signals to the FPGA. Pull-up resistors of value 1 kΩ are used because port 4 has open-collector drivers on it. P5.2 and P5.1 are cleared and P5.0 is set to enable MMIO for the bus expansion to the FPGA. See MicroPac 535 manual page 7 for more complete detail on how to initialize the μC.
Once the µC is properly initialized, the FPGA is taken out of reset (P4.4 set), put into calibration mode (P4.5 set), and the µC enters a loop to wait for the limit interrupt. More detail about the calibrate cycle is given in the FPGA Design section. The limit signal triggers EX3. The EX3 ISR takes the FPGA out of calibrate mode (P4.5 cleared) and disables EX3 so that the interrupt is not triggered in bore sight tracking mode. As discussed in the “FPGA Design” section, when the limit signal is high, the AT register is set to “C3h”. However once the FPGA is taken out of calibration mode, the µC software does not allow it to move past the “C2h” position.

The FPGA can be reset and calibrated without restarting the program via the keypad. The EMAC board is equipped with a sixteen key keypad. When a key is pressed, external interrupt 1 is triggered. The keypad module decodes the number sent by the keypad and performs the appropriate operations. The MicroPac 535 manual contains more detailed information on how the keypad functions.

The communications module handles the serial communication between the PC and the µC and the MMIO communication between the µC and the FPGA. The EMAC board is equipped with 3 serial communication ports: COM0, COM1, and COM2. COM0 is internal to the 80535 µC and is used by the Keil monitor software. COM’s 1 and 2 are on an external DUART, an SC26C92 chip. COM2 is being used for the communication between the PC and the µC. The DUART is set-up to run 8 data bits, 1 stop bit, no parity at a rate of 9600 baud. The MicroPac 535 and SC26C92 manuals contain more detailed information on how to initialize the DUART. The data that is sent by the PC is a two’s-complement number of steps the stepper motor needs to turn to center the image in the camera’s field of view. A positive number turns the camera clockwise and a negative number counter clockwise. Once the number is received from the PC, the current camera position is read off the FPGA. Adding the change in position to the current position results in the desired new position of the camera. The new position is then written to the FPGA. The communication routine will be run every one second. This time is set by the IP software on the PC, which has a refresh rate of approximately 1 second.

Figure 13 shows the communication-wiring diagram for the system. The PC is connected to the µC board via a serial cable. The µC communicates with the FPGA board via three wire bundles indicated in the figure by the ellipses. The bundles are made with 22-gauge solid core copper wire and have insulation colors as indicated. The two 1 kΩ pull-up resistors are needed because port 4 on the EMAC board has an open-collector driver IC on it. The open-collector chip can sink 0.5 A. The 1 kΩ resistors allow 5 mA of current. Also indicated in the figure, are the wires for the 1 MHz clock and the limit signals.

Appendix A contains flowcharts for all micro-controller software.
Design Obstacles

While designing the system, two problems were encountered: VHDL design software compatibility and FPGA implementation limitations.

The initial VHDL design of the camera controller was originally done with Model Sim on a Sun station. Model Sim is an excellent coding and simulation tool for VHDL applications. Another UNIX-based software package is needed to synthesize the VHDL design. Leonardo Spectrum (LS) was used here. Once the design has been synthesized, a .bit file needs to be created in order to program the FPGA. The LS software creates a .edf file which can be translated and mapped into a .bit file. However, there is no software package on the Sun machines that will do this. The needed software is on the Windows-based computers. This software package is called Xilinx Foundation (XF). The XF software is supposed to be able to make a .bit file from a .edf file created by another program. This does not happen successfully, however. There is an incompatibility somewhere that causes the XF to return erroneous errors. This incompatibility could not be resolved; therefore, the solution was to do all VHDL design with the XF software package.
The original controller entity design required a counter (AT register) with both asynchronous reset and preset, the reset for the global FPGA reset and the preset for loading the calibration value when the limit signal was high. The XC4000 series FPGA, however, does not support this feature. The design could have been switched over to a XC9500 series CPLD, but it was not known that these were available at the time. The solution of this was the loading of the counter with the calibration value before the camera actually reached calibrated position. Therefore, the camera position is invalid until the camera reaches the calibration position. This was an acceptable situation.

**Vehicle Control**

The Vehicle Control system is divided into two main components the steering control and the drive control systems. The purpose of the drive control system is to control the speed of the vehicle while it is intercepting the target. The purpose of the steering control is to minimize the angle between the camera bore sight and the vehicle bore sight so the vehicle stays on track to intercept the target. Both of the systems are controlled by the micro-controller and are powered by two 12-volt DC batteries, which are fused at 30 amps for safety. One battery is used to power the micro-controller and power electronics and the other powers the drive motors. A general block diagram of the vehicle control system is shown in Figure 14.

![Vehicle Control System Diagram](image-url)

*Figure 14, Sub-system Block Diagram of the Vehicle Control*
Drive Control System

The drive motors are driven by a pulse width modulated (PWM) signal generated by the micro-controller using Timer 2. For initialization code for Timer 2 see MicroPac 535 and 80515 manuals. The PWM signal is sent from P1.2 on the digital I/O into two parallel Mosfet transistors (IRFP240), that turn on and off according to the duty cycle being sent. The drive motor control system is optically isolated from the micro-controller by a 4N25 optical isolator. This is done to prevent large voltage spikes that occur when the motor switches on and off, from getting back to the micro-controller. The voltage spikes are due to residual current in the system that has not been fully dissipated. There is also a freewheeling (1N1183A) diode in the system to help dissipate the excess current so the large voltage spikes does not damage the Mosfet transistors. The power circuitry for the drive motors can be seen in Figure 15, the circuitry consists of the parallel Mosfet transistors and the freewheeling diode.

The speed is controlled by the keypad routine, which uses ext1srv (external interrupt 1 service). The keypad is set up, so that if buttons 4,5,6,7 are pushed, the corresponding duty cycle is sent to P1.2. This was done so that an optimum speed could be found at which the image processing could keep up. If the vehicle travels too fast for the image processing, the target would be easily lost. If the vehicle travels too slowly, it would not be able to intercept the target. It was determined that 75%-80% duty cycle was needed to be sent to the vehicles drive motors.
There were a few reasons for this. First, due to the weight of the vehicle that large of a duty cycle was need to over come static friction. The other reason was the battery supplying power to the drive motors was faulty and was not producing enough current at low speeds to drive the motors.

**Steering Control System**

For the steering control system the micro-controller receives the current position of the camera from the FPGA and then using a lookup table determines the correct position for the wheels so that the camera bore sight is aligned with the vehicle bore sight. Since the camera has a greater turning range then the car, (00h-C2h=194 values for the camera to 71h-FFh=142 values for the car). The lookup table had to be designed so for every position of the camera there was a one to one ratio with the steering wheels. So the first step was to make the center position of the camera the same as that of the wheels. So the center position of the camera is the 97th value so B8 hex, center position of the wheels, was placed in the 97th position of the table. Also since the camera has a greater turning radius then the car only ten degrees of accuracy was needed on either side of center. After the ten degrees the wheels on the vehicle would turn full left or full right.

Once the lookup table had determined what position the wheels need to be, to align the bore sights, that value was place in a variable called POS. Next the TMR0SRV module would then take the value stored in POS and compare that to the actual position of the steering wheels. The actual position of the steering wheels is determined by the feedback from the linear actuator. The feedback from the linear actuator is feed into the A/D of the micro-controller via analog channel 0 on HDR4. The code for the A/D module can be seen on the website. The voltage range entering the A/D is roughly between 2.5 and 5 volts, which corresponds to a binary range of 71h to FFh. If the value from the A/D is less then the value in POS the vehicle turns right and vice versa if the A/D is greater then the value in POS the vehicle turns left. Once the value of the A/D and POS are equal the vehicle stops turning. The TMR0SRV module code can be seen on the website.

The linear actuator is what physically turns the wheels of the vehicle. The H-bridge on board the vehicle allows the process of controlling the linear actuator to be simple. The micro controller sends three signals to the H-bridge, direction, brake, and a PWM. The H-bridge pinouts can be seen in Figure 16. The PWM controls the rate at which the wheels turn. Like in the speed control system the PWM is generated by timer 2 and is inputted into pin 5 on the H-bridge from P1.1 on the digital I/O. It was determined a 75% PWM signal produced the desired rotating speed. The direction control is handled by simply toggling P4.0 on the digital I/O; from there it is inputted into pin 3 on the h-bridge. When P4.0 is set high the wheels turn left and when it is set low the wheels turn right. The brake is used to stop the wheels from turning. The brake is set by making P4.1 on the digital I/O go high which causes pin 4 on the H-bridge to go high thus setting the brake and not allowing the wheels to turn in either direction.
Implementation, Results, and Evaluation

Image Processing

The image processing was made to be as fast as possible and as accurate as possible. The final iteration for the image processing is a compromise between the two. The sample time of the image processing is roughly one second. This allowed time for the hardware to align the camera with the object before the next picture is taken so the image captured is not blurry.

The image processing works best when the target contrasts with the background. The more vivid the target is against the background, the more likely it will be detected. The noisier the background is, the less likely the image processing will detect the object. A noisy background is defined as a background that has colors that are not contrasting with the target. If the reference color were a light red, for example, a noisy background would be a brown table. This is because a brown table and a light red object have close to the same RGB vector. Other examples are a red door thirty yards away.

Another possible problem with the image processing is the brightness of the image. Since the camera auto adjusts for the brightness, any large change in lighting will cause the target’s color value to change with respect to the reference pixel’s value. The other danger with a lot of ambient light is the colors will be ‘washed out’. This means the colors will appear very light or even white. This causes all the colors to have very close to the same RGB vectors. When this happens the image processing cannot differentiate between the background and the object.
The best way to tell if the image processing is going to work is to look at the image the camera sees. If this image appears very bright the image processing might not work. If the background contains a lot of color close to the reference color the image processing might track that and not the object. The more contrast between the object and the background the better it works.

One special case of where the image processing will not work is if the image is gray scale. At this point, the RGB vector components are all the same value. When the reference pixel is gray the transform results in the same line. So any color of gray between white and black will be detected as the object. The placement of the reference pixel is seen in Figure 17. The further away from the gray line the color is, the better the image processing will detect it.

Figure 17. The further away from any points where the components of the RGB vector are equal, the better the image processing will work and the more tolerant it will be to noise.

The object size is also something that must be considered. The larger the object is the more pixels will be correlated with the reference pixel. This allows for more noise in the background without suffering from tracking the noise instead of the object. The trade-off between object size and background noise is the thing that needs to be looked at the most when trying to detect the object. Washed out color is considered noise, as is color close to the reference color. The larger the object is in the camera view the brighter the lighting can be.
Camera Control

The implementation of the camera controller was not as straightforward as expected. A couple of obstacles impeded the progress of implementation: FPGA expansion board pin conflicts and non-functioning FPGA pins.

The XS40 FPGA expansion board has a small µC and an SDRAM chip on it. These chips cause undesired results when interfacing with an external µC with bus expansion. If the pins that either of the chips are used in the interface, the chips cause unacceptable loading conditions, which, in the case of the EMAC board, cause the µC to be put into a constant reset state. This problem was remedied by moving all of the data lines to pins on the FPGA board that do not have any other active components on them. Such pins are the ones that are connected to the seven-segment display. Another solution, which was implemented but not tested, is to disable the SDRAM and hold the µC in reset.

Moving the data lines solved the continuous EMAC reset problem, but data was still not being sent to the FPGA correctly. The cause of this problem turned out to be non-functioning pins on the FPGA to which the bus expansion address lines were connected. Even though the pins were configured to be inputs, they were constantly driving high. The pins were tested by attempting to pull the inputs low with a 1 kΩ resistor. When pins continued to drive high, it was determined that the pins were non-functional.

Once the problems were fixed, however, the implementation of the camera control system was straightforward. The serial communication was tested first.

The serial communication module was tested by connecting the µC to a PC with a hyper terminal running with the proper serial protocol. ASCII characters were sent to the µC from the PC and displayed on the EMAC board LCD display. At this point a glitch was found in the serial communication. When the same character was repeatedly sent, every other character received would be incorrect. To see the actual bit values being sent, a routine was made that displayed the 8-bit sequence read from the DUART on the LCD. This revealed that the most significant bit was toggling between 0 and 1 every time the same character was sent consecutively. Due to time constraints, a extensive investigation was not made in order to find the cause of the glitch. Instead, a quick fix was utilized. Since the command sent by the image processing would never be greater than ±11, bit 6 and bit 7 of the command byte would always the same value. Thus, the problem was fixed by simply coping the value of bit 6 into bit 7 after every transfer.
The communication with the FPGA was tested by creating a keypad routine that moved the camera to different positions for given key presses. The keys used and their actions are as follows:

- F → Move camera full CW
- 0 → Move camera full CCW
- C → Move camera to center
- A → Move camera N steps CW
- B → Move camera N steps CCW
- 1 → Set N = 1
- 2 → Set N = 2
- 3 → Set N = 3
- 4 → Set N = 4
- 5 → Set N = 5
- E → Calibrate the motor
- D → Toggle the reset of the FPGA

The increment functions were set up to test if the µC could read from the FPGA. This is tested because the µC has to read the current position from the FPGA and add or subtract N from that. This test was successful.

Next the serial and bus expansion modules were tested together. Again, a Hyper Terminal on the PC was used to send serial data to the µC. This data was then added to the camera's current position and sent back to the FPGA. The camera position was successfully controlled by the PC.

**Vehicle Control**

The vehicle control system was successfully implemented. It was a straightforward design once certain things had been cleared up. The first was the power electronic, what signals need to be sent to what in order to get the drive wheels to turn and the linear actuator to move to turn the steering wheels. There was also a learning curve time to learn how the H-bridge controls the linear actuator. They only real problems that were in counter were faulty components. But that was mostly due to human error of hooking the power cable in, in reverse polarity. This tested our trouble shooting skills. These problems were simply solved by using a digital scope to test the inputs and outputs of the components to make sure they had the correct signal entering and exiting. For example when testing the H-bridge, the input pins for the direction, brake, and PWM were looked at. Those were correct but when checking the output pins of the H-bridge, it was discovered that the output was incorrect for the input signals and the output was also incredibly noise. This indicated the H-bridge had gone bad and need to be replaced.

The first step in testing the vehicle control system was to test it separately from the IP and camera control system. This was done setting up a keypad routine that would simulate, changing position commands like those it would receive from the camera control system. The keypad routine also tested the speed control by having different keys on the keypad correspond to different PWM signals. After it was successfully tested, the next step was to integrate the camera control with vehicle control.
The keypad was used to simulate position signal, but this time it was simulating position signals that would be sent from the image processing. When the camera would rotate the wheels on the vehicle rotated to minimize the angle between the wheels and the camera to align the two bore sights. Once that had been successfully tested the final step was to integrate the image process with the camera control and the vehicle control. The image processing sent the position signal corresponding to the position of the target. So the camera control system would rotate to keep the bore sight of the camera aligned on the target. Then the steering wheels would rotate to again minimize the angle between the camera bore sight and vehicle bore sight. The keypad was used to control the vehicle speed, it was discovered that the vehicle needed travel at low rated in order for the image processing to keep the target in sight.

Conclusions

The goal of the project was to design an autonomous vehicle that would track and intercept a moving target. The final system accomplished this task in a very controlled environment. If, however, the lighting conditions varied too greatly or the background contained colors too close to the target color, the vehicle would lock onto false targets and not track the desired target.

Additional Resources

In addition to this report, a website contained all the code is on the web. It also includes pictures and other materials related to the project. The web page is at:

Http://cegt201.bradley.edu/projects/proj2002/homers/index.html

A CD-ROM with all source code, the project final presentation, and documentation pictures is also available.
Appendix A
Flowcharts for Micro-controller Software

Start
EMAC Init
FPGA Init
Serial Init
T0 Init
T2 Init

T0 ISR
Check Vehicle State
Vehicle State Correct
N
Adjust Vehicle State
RETI

Serial ISR
Read Data from Buffer
Read From FPGA
Write Data to FPGA
RETI

Keypad ISR
Decode Key Value
Perform Proper Action
RETI
Appendix A
Continued

T0 Init
- Set T0 to 16-bit Timer w/ Auto-reload
  - Set Reload Values to Run at 2 ms
    - RETI

T2 Init
- Set Compare And Reload Values
  - Set Timer Compare Mode 0
    - RETI

EX3 ISR
- Exit Calibration Mode
  - Disable EX3
    - Center Camera
      - RETI