Mudd Rover

☯Design Document☯
Version 0.2

Ron Coleman Josef Brks Schenker Akshay Kumar Athena Ledakis Justin Titi

1. Overview

The basic idea behind Mudd Rover is to create an autonomous robot with an onboard camera that is capable of finding a line on the floor, orienting itself properly and then finally following it. The processing done behind the scenes will take place using the XiLinx Spartan FPGA that is mounted on XSB-300E. Part of the FPGA will be programmed to be our custom video processing hardware. The skeletal form of the robot will be a tank-like vehicle built with Legos™. The method of communication between the XSB-300E and the Mudd Rover will take advantage of the Lego RCX and its companion serial IR Tower. Primitive commands will be sent from the XSB-300E via the serial IR tower to the Lego RCX, which will be mounted on the Mudd Rover.

The seamless communication between the XSB-300E and the Mudd Rover will give the appearance of an intelligent robot with a vast amount of processing power onboard even though 100% of the processing will actually be taking place remotely. There are three distinct parts to this project: Video Processing, Serial / IR Communication, and the Brain of the Mudd Rover.

2. Video Processing

2.1. Overview

The X10 Video camera will be connected to the SAA7114 Video Decoder located on the XSB-300E via a composite video cable. The X10 video camera will be sending a constant NTSC signal to the SAA7114 decoder. The SAA7114 decoder will process this stream of video and send it out in digital form to our custom video processor and its accompanying software. This package of hardware and software will be refered to as SourceFinder from this point on. SourceFinder will then process select frames of the digitized video and gather the information needed by the Brain of the Mudd Rover in order to direct Mudd Rover to achieve its task.

2.2. SAA7114 Video Decoder

The first step in our video processing is the Phillips SA7114H video decoder. The video input direct from the camera mounted on the Mudd Rover to the chip via the RCA jacks (either J7 or J8 on the board), which connect to AI23 and AI24 respectively. We will deal with the raw data in the YUV 4:2:2: format. For the purposes of our project, however, we will only be using the Luminance, value, Y—so practically we will only use every other bit inputted into the system. Rather than using the Phillips chip for any processing, we simply use it as an analog-digital converter. The data enters the chip in an 8 bit word, and gets passed straight to SourceFinder.

2.3. SourceFinder-Hardware

The hardware component of SourceFinder is a simple component in VHDL that takes the video input one line at a time and outputs the location and size of the largest light source on the line. As inputs, SourceFinder takes the 8 bit data stream from the SAA7114 Video Decoder, the video clock, and the Horizontal and Vertical reference values. Using these inputs, the hardware runs through each line building the largest light source and outputs it for use by the software component of SourceFinder. SourceFinder outputs two pieces of information, the position of the light source (using a 10 bit wire) and the length of the source—also a 10 bit integer. Since the SAA7114 Video Decoder chip reads video at 30 frames a second, by performing this task in hardware, we can read new inputs at 5-6 frames a second with great enough speed to ensure that our programs can react in real time to any input offered.

2.4. SourceFinder-Software

2.4.1. Overview

Upon leaving the SourceFinder Hardware component, position and length of the light source are sent to the software component of SourceFinder. It reads in the values and thereby determines the location of the total light source on the entire screen. The goal of this program is to find the central point of a distinguished area on the path that the rover will be following. We will also determine the direction that the rover will be moving in, by finding the slope of the path in each given frame.

This is done in 4 steps:

1. Find the central point of each line.
2. Divide each frame into equal sections and find the center of each section of the frame, using information from step 1.
3. Using information from step 2, find the central point of each frame.
4. Find the slope of the line, by calculating the line between the top and bottom section of each frame.

2.4.2. Implementation

The input for this program is the start point and the length for a distinguished (either dark or light) set of pixels for one line of a given frame. The algorithm we will use is:

```
#define SCREENSECTIONS 2
#define LINES_PER_SEC 5
#define FRAMEHEIGHT SCREENSECTIONS * LINES_PER_SEC

int startposition;
int length;
int i = 0;
int s = 0;
int j = 0;
int main(){
        int lineaverage[LINES_PER_SEC];
        int sectioncenter[SCREENSECTIONS];
        int lineslope;
        int tempsum = 0;
        for( , , )
        {/*NOT REAL CODE
        startposition = getstartposition();
        length = getlength();
        */
                lineaverage[i] = startposition + (length/2);//find out how to shift to divide
                i++;
                //finds center of each line
                if(i == LINES_PER_SEC){
                        i=0;
                        for(j=0; j<LINES_PER_SEC; j++){
                        sumofaverages = lineaverage[j] + sumofaverages;
                        }//end for
                        sectioncenter[s] = sumofaverages / LINES_PER_SEC;
                        sumofaverages = 0;
                        s++;
                        if(s == SCREENSECTIONS){
                                s = 0;
```

```
                                        for(j=0; j < SCREENSECTIONS; j++){
                                                tempsum = sectioncenter[j] + tempsum;
                                        }// end for
                                        centerpoint = tempsum / SCREENSECTIONS;
                                        tempsum = 0;
                                        lineslope = (LINES_PER_SECTION * (SCREENSECTIONS - 1) ) /
(sectioncenter[0] - sectioncenter[SCREENSECTIONS - 1]);
                                                                        // This rise can be hard coded when we
know the exact specs
                                }//end if
                        }//end if
                }//end for
}// end main
```

2.4.3. Performance and Memory Usage

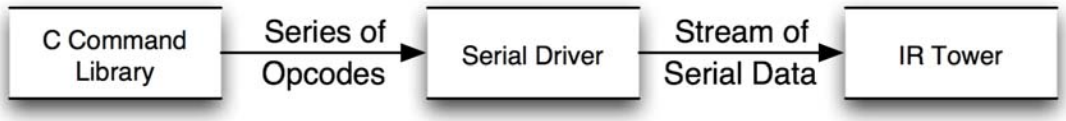
The estimated runtime of this algorithm is O(2(N+k)), where N is the number of lines per frame and k is the number of sections that each frame is split up into. Assuming we will split the screen into about 6-8 sections, k is less than 3% of N, so its addition to the runtime is minimal, but we are including it for good measure.

In order to run this program we will be storing 7 variables of type int, which would require 14 bytes of storage. In addition we will be storing N/k + k variables of type int. This would require 2*(N/k + k) bytes for storage. N is the number of lines per frame and k is the number of sections that we decide to divide a screen into.

3. Communication

3.1. Overview

At the core of the communication between the XSB-300E and the Mudd Rover are two components. The first of these components is the serial driver. The serial driver will take a hexadecimal opcode and transmit it over the serial cable to the IR tower. The second component is a small C library that will bunch these opcodes together to simplify and facilitate moving the robot from the main C program (Mudd Rover Brain).



3.2. Opcode Basics

An opcode is 2 bytes in length. From this point on these bytes will be referenced using hexadecimal with a slash separating each byte. Below is a list of the supported opcodes, which is only a small subset of the full library of opcodes supported by the RCX:

Lego RCX Opcode	Hex Value	Description
Play Sound	51/59	Play specified sound(for debugging)
Set Motor Direction	E1/E9	Set the direction of specified motors
Set Motor On / Off	21/29	Set the on/off state of the motors accordingly
Set Motor Speed	13/1B	Set the speed of the motors accordingly
Set Transmitter Range	31/39	Set the transmitter range accordingly

3.3. Opcode Format for Transmission

A 3-byte header that is the same for all opcodes sent to the IR tower must precede each opcode. Depending upon the opcode being sent, it can also be accompanied by several bytes of data after the opcode. With this in mind a complete opcode will look like this:

Message Header(3 Bytes)	Opcode(2 Bytes)	Opcode Arguments(At Most 3 Bytes)

3.4. Opcode Details

3.4.1. Play Sound

Opcode:51/59
Arguments: byte *sound*

Sound	
Index	Description
0	Blip
1	Beep Beep
2	Downward Tones
3	Upward Tones
4	Low Buzz
5	Fast Upward Tones

3.4.2. Set Motor Direction

Opcode:E1/E9
Arguments: byte *code*

Code	
Bit	Description
0x01	Modify Direction of motor A
0x02	Modify Direction of motor B
0x04	Modify Direction of motor C
0x40	Flip the direction of the specified motors
0x80	Set the directions of the specified motors

3.4.3. Set Motor On / Off
Opcode: 21/29
Arguments: byte *code*

Code	
Bit	Description
0x01	Modify On / Off state of motor A
0x02	Modify On / Off state of motor B
0x04	Modify On / Off state of motor C
0x40	Turn off the specified motors
0x80	Turn on the specified motors

3.4.4. Set Motor Speed
Opcode: 13/1B
Arguments: byte *motors*, byte *source*, byte *argument*

Motors	
Bit	Description
0x01	Modify power level of motor A
0x02	Modify power level of motor B
0x04	Modify power level of motor C

Source specifies the source type for power level. It can only take on values of 0, 2, and 4.

Argument specifies a value from 0-7 for the power of the motor, with 7 being the fastest.

3.4.5. Set Transmitter Range
Opcode: 31/39
Arguments: byte *range*

Range sets the transmitter to short range when 0 and long range when 1

3.5. Serial Communication

3.5.1. Protocol

Each opcode is sent via serial to the IR tower at with the following specification:

Baud Rate	2400
Non-Return to Zero	Yes
Stop Bit	1

Start Bit	1
Parity	Odd

3.6. Implementation

3.6.1. Overview

The serial communications package will be written in C and be comprised of two components, a serial driver and a library of C functions.

3.6.1.1.Library of C Functions

The functions are very simple but useful groupings of opcodes to achieve basic actions for the robot. Below are the planned library functions:

Function	Description
TurnLeft(int speed)	Turns the robot left at a specified speed
TurnRight(int speed)	Turns Mudd Rover right at a specified speed
Forward(int speed)	Moves Mudd Rover forward at a specified speed
Backward(int speed)	Moves Mudd Rover backward at a specified speed

3.6.1.2.Serial Driver

The serial driver will take an opcode and accompanying bytes and transmit them over serial to the IR tower using the proper format and protocol as specified in earlier sections. The primary goal is to have a serial driver the sends data but if time permits we would also like to have it receive data as this will enrich the robustness of the robot and we will also be able to gather data in addition to video.

4. Brains of Mudd Rover

4.1. Overview

Now that the hardware and software groundwork for both the input and output of the project has been laid out it is now appropriate to talk about the Brains of Mudd Rover. The Brain component of Mudd Rover will act as a mediator between the input and the output of the project. It will take the information given to it from SourceFinder, decide the appropriate actions to be taken, and then send out commands to the Mudd Rover.

4.2. Implementation

This part of the project will very much be one that evolves as the project moves on. However there are some key goals that the Brain must achieve in order to reach the overall goal of line following. The pseudo code for the infinite loop is as follows:

1. Find Line
    a. If you can't find the line, drive around in increasing concentric circles until you find one.
2. Orient the Rover to be going in the correct direction of the line
    a. If not oriented, stop and turn accordingly
3. Follow line by keeping it centered in the frame
    a. If not centered adjust the speed of the appropriate motor to create a small arc to get back on course
    b. Repeat infinitely while line is still visible, if not go back to step 1