CMPE 490: Progress Report

Automated Life Preserver Launcher

Curtis Sand, curtissand@gmail.com

Zach Byrne, zbyrne@ualberta.ca

Preferred lab sections: Monday, Tuesday, Wednesday lab sections

Declaration of Original Content

The design elements of this project and report ore entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

The Point of Interest algorithm for finding the target in the picture was suggested by Dr. Elliott.

[1] The triangulation method described in the Design and Description of Operations Section.

[6] The Schematics for the Stepper Motor Control Circuit was provided by the L297 Data-sheet.

Abstract

When you're a fisherman off the coast of Labrador or a passenger on a Caribbean cruise there is always a chance of falling overboard. In a man overboard situation the two main things that another crew member must attempt to do are keep a finger pointed at the victim and to get a life preserver to the victim. We propose an automated system designed to locate a man overboard and accurately fire a life preserver to them. The Automated Life Preserver Launcher will consist of three functional modules: a camera system, a micro-controller, and a life preserver launching turret. The camera system will consist of two cameras that will be used to locate an infra-red light source. The micro-controller will then calculate the distance to the object using a binocular range finding algorithm and calculate the required trajectory for the turret. The turret module will use targeting data from the micro-controller to aim and then fire the life preserver projectile. Some assumptions or simplifications that will be made are that the target is stationary, a bright white LED can be substituted for a human infra-red light source and a toy Nerf gun can stand in for an actual life preserver launcher. Thus far we have successfully written code to communicate with the cameras and we've wired and successfully tested the stepper motor control circuits. Also, we have proven that the range finding algorithm works and is fairly accurate as long as the physical parameters of the system are known.

Table of Contents

Functional Requirements and Optional Extensions

For the Automated Life Preserver Launcher to be considered a 100% successful project we would expect the following functionality: A button is pressed to alert the machine of an overboard crew member and the system automatically acquires the infra-red target, aims and accurately fires a life saving flotation device at the location of the crew member. Due to scope issues and availability of mechanical parts, we will be making a few simplifications and two very important assumptions to make a proof-of-concept prototype of this system attainable.

The first of the two assumptions that will be made is that the target is stationary. Motion tracking, and compensation for motion in targeting would be a requirement if this were to be mounted on a boat; however, we are attempting to show that a system that uses an infra-red beacon as a target can calculate the distance of the target, determine an appropriate ballistic trajectory and fire a projectile with reasonable accuracy. The second assumption that we will make is that the infra-red beacon that we will use is the brightest infra-red source within the view of the camera.

Some simplifications in the setup that we will use include using a toy Nerf gun to stand in for a life preserver launcher, an LED to stand in for a human infra-red source, and that targeting time is not an issue. We would like to show that it is possible to use pictures from two parallel facing cameras to calculate the distance of an object and to subsequently calculate the appropriate trajectory for a projectile to hit the target.

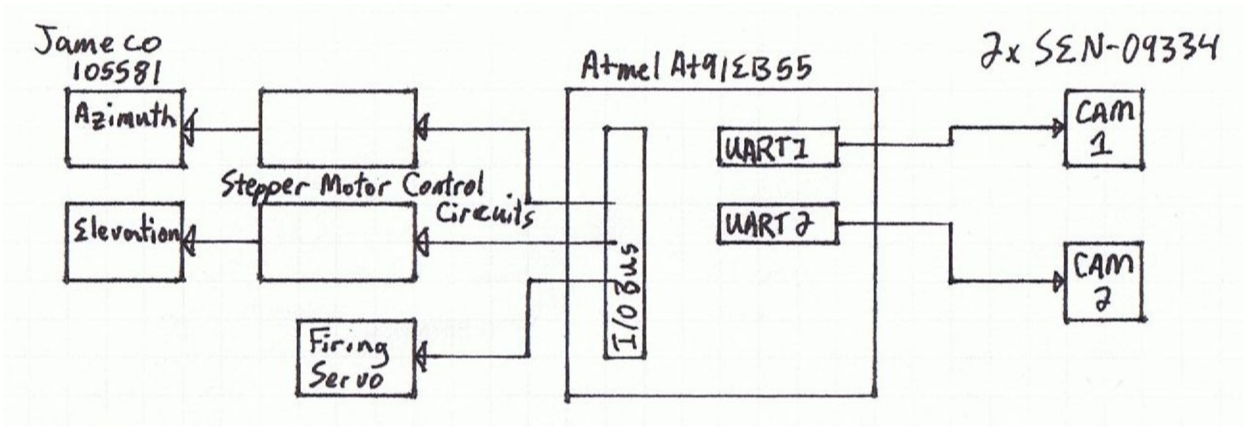
One thing that we were not very sure about was whether or not we could use an infra-red light source as our target. It seems, however, that the camera that we have ordered has an IR Filter built in. Depending on the pass bandwidth of the filter we may be required to implement a backup plan and use a bright white LED instead of an IR LED as a target.

Another problem that we have also considered is the amount of work that may be required to interface the micro-controller with the two cameras. This work may impede integration of other parts of our project. As a preventative measure we have split the project up into essentially three functional sections (see the Design and Description of Operation section for more information.). In this way the two of us can develop each section in parallel and show working functionality in each section separately if need be.

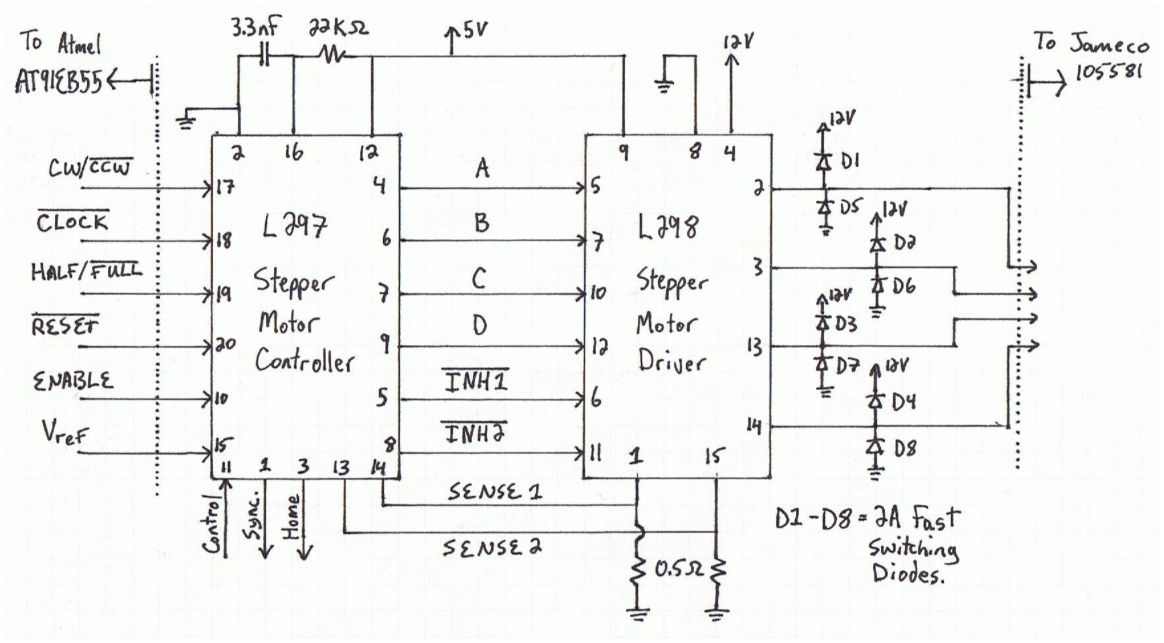
Additions and extensions that could be added to this are many. Due to the use of infra-red light for targeting this type of system could be used to fight fires, or track vehicles with a video camera. An obvious next step from using an infra-red beacon and and standard digital camera would be to use an infra-red camera and image detection to recognize a human. As mentioned earlier, motion tracking would be a logical next step for a system such as this. Another extension that could be added would be an actual life preserver launcher with a cable and a winch for retrieving the target and multiple target recognition and prioritization so that a multiple man-overboard situation could be dealt with.

Design and Description of Operation

The Automated Life Preserver Launcher (ALPL) will be broken down into three functional modules or sections. The first section is the interface between the two cameras and the micro-controller The goal of this section is to successfully acquire a single frame from each camera and store it temporarily in RAM on the micro-controller. The second section is a software section that will analyze the two images from the cameras and identify the infra-red target, calculate the distance to the target and then, based on the location of the target, calculate the appropriate angle to aim the turret. The third section will be the turret section which will take a digital signal, decode the signal and then adjust the turret to the appropriate trajectory.



The ALPL System will use the Atmel AT91EB55 micro-controller as its brains. Connected to two of the UARTs on the Atmel micro-controller will be a SEN-09334 JPEG Color Camera. An issue with the UART that we have come across is that the 1st and 2nd UART (UART 0 and 1) is on the same RS232 chip as the Angel Debugger. This means that in order to interface our cameras we'll need to use the slingshot to program the Atmel board. A downfall of using the slingshot to program the micro-controller is that you loose the ability to debug interrupts. To get around this we will program and debug our interrupts using the Angel debugger and only using one camera on UART 2. Finally connected to the IO bus of the Atmel micro-controller will be 2 Stepper Motor Control Circuits and a servo. The control circuits (shown below) take input from the micro-controller and drive the stepper motors. For a full size view of the Stepper Motor Control Circuit see the Schematics section at the end.



In order for the system to accurately calculate the distance of the target, the two cameras will need to be securely mounted so that they face in parallel directions. In this way the difference in the location of the target in the images from the two cameras can be used to calculate both the horizontal position and the distance of the target. The centroid of the light source is calculated by going through the pixels of the image one by one and finding the average coordinates of all pixel locations with a brightness above a threshold value. In the formula's below d is the distance to the target. Offset is the number of pixels from center that the target is found in the pictures. P is a constant with units of Degrees per Pixel. P is found by dividing the Field of View of the camera by the number of pixels in the picture. This gives us a very rough measure of the real world from the camera. From the data-sheet of the SEN-09334 the Field of View is 57 degrees diagonally and the resolution of the picture is 640x480pixels. This gives us P=0.07. In general the number of degrees per pixel is not constant across the entire Field of View but we hypothesize that it will be accurate enough for our purposes. A more complex model can be considered if this hypothesis is proven wrong.

$$d=\frac{l}{(\frac{1}{\tan(a)})+(\frac{1}{\tan(b)})} \quad \text{where} \quad a,b=90-(offset\times P)$$

We will need to do a series of experiments to build a lookup table that can be used to translate the distance to the target into the required angle that is needed to hit the target. The values in the table will be determined from firing the Nerf gun from the turret at different angles. If linear interpolation of the table values turns out to be insufficient, a more complex interpolation method may be used, e.g. a cubic spline. A requirement of the mechanical turret is that its range of motion be sufficient that the angle required to hit the target for any distance between the range of the Nerf gun and the minimum range of the binocular camera system.

The design of the turret is a simple two axis design so that there will be both azimuth and elevation articulation. The turret's initial position will be manually configured before each firing using a purpose written subroutine. Time permitting, we would like to add a photo-sensor connected to the 'Home' input to the L297 Stepper Motor and to generate an interrupt to tell the micro-controller that the turret is in it's home position. To fire the Nerf gun programatically we originally thought of using a solenoid. In looking at the available parts sites, we found that solenoids were very expensive for such a simple operation. As a backup plan we have determined that we can use a servo in such a way that when the servo turns an arm is pressed against the trigger.

Hardware Requirements

To construct the ALPL system we will need a micro-controller (processor speed has been made irrelevant by our stationary target assumption), two stepper motors and a hobby servo for the turret, 2 digital cameras and a piece of interfacing hardware to drive the stepper motors on the turret.

The micro-controller that we plan to use is the Atmel AT91EB55 board that is available in the lab. The cameras that we plan to use will connect to the Atmel board via two of the available UARTs. The stepper motors that drive the turret will be connected to a circuit that translates a clock signal, a direction signal and an enable signal into the appropriate control signals for the stepper motor.

The camera model that we are planning to use can output a single JPEG frame at a time at 640x480 resolution. In the standard JPEG encoding there are three bytes of information for each pixel which boils down to approx. 900kb per image at maximum resolution. Considering that the Atmel board has about 128Kb of space for user code and that we will also need space for a second image, we will need to interface some ram with the micro-controller We will most likely use the lowest resolution available on our cameras but this gives an estimate to our space requirements. As an educated guess, we can say that we will need at maximum 2000Kb of RAM.

Parts List

- 2 digital cameras: C328-7640(sparkfun:SEN-09334) JPEG Color Camera with a UART Interface
  - An order sheet was submitted for approval before the proposal deadline.
  - $54.95US from sparkfun.com
  - data-sheet: http://www.sparkfun.com/datasheets/Sensors/Imaging/C328.pdf
  - manual: http://www.sparkfun.com/datasheets/Sensors/Imaging/C328_UM.pdf
  - requires some parts (from digikey) for making connections:
    - WM-8289-ND -- 2mm connection Receptacle housing (4 position female) http://parts.digikey.com/1/parts/1210042-conn-rcpt-hsng-2mm-4pos-single-51090-0400.html
      - $0.33 from digikey.com
    - WM-6050-ND -- crimps, female 24-30AWG, 2mm http://parts.digikey.com/1/parts/271044-conn-term-female-24-30awg-tin-50212-8100.html
      - $3.54/10units from digikey.com
- 2 Stepper Motors
  - Jameco 105581 12V Stepper Motor
  - In stock at the school
  - The Data Sheet is no longer available online so see Nancy Minderman.
- 1 servo
  - After talking to Nancy Minderman the recommended servos are either HS422HB or HS635HB by Hitec. These servos are in stock at the school. We will do an experiment to see which servo has the required torque to pull the trigger of the Nerf gun.
  - Hitec HS422:
    - data-sheet: http://www.robotshop.ca/PDF/hs422.pdf
    - User Manual: http://www.robotshop.ca/PDF/Servomanual.pdf
  - Hitec HS635HB:
    - data-sheet: http://www.hitecrcd.com/product_file/file/56/HS635.pdf
    - User Manual: http://www.hitecrcd.com/product_file/file/55/Servomanual.pdf
- L297 Stepper Motor Controller
  - In stock at the School
  - Data-sheet: http://www.st.com/stonline/books/pdf/docs/1334.pdf
- L298 Stepper Motor Driver
  - In stock at the school
  - Data-sheet: http://www.st.com/stonline/books/pdf/docs/1773.pdf

Table of User IO Signals

Camera1-TxD	ARM-PA15/TXD0
Camera1-RxD	ARM-PA16/RXD0
Camera2-TxD	ARM-PA21TXD2
Camera2-RxD	ARM-PA22RXD2
Stepper1-CW/CCW	ARM-PB8
Stepper1-Clock	ARM-PB20/TIOA0
Stepper1-Half/Full	ARM-PB9
Stepper1-Reset	ARM-PB10
Stepper1-Enable	ARM-PB11
Stepper2-CW/CCW	ARM-PB12
Stepper2-Clock	ARM-PB23/TIOA1
Stepper2-Half/Full	ARM-PB13
Stepper2-Reset	ARM-PB14
Stepper2-Enable	ARM-PB15
Servo-Control	ARM-PB26/TIOA2

Software Design

The software system for our project is broken into several modules, starting with a main function and branching out into the modules performing various tasks. The main function will set up interrupts, initialize I/O pins and calls init_motors() which configures the stepper motors to a known state. After this, main() calls start(). Start waits for the start button to be pressed, this will be signaled by an interrupt modifying a semaphore or similar data structure. At this point, we will be applying preprocessor directives to develop different levels of complexity as our design progresses. The first step will simply use buttons to manually position the stepper motors and fire the Nerf dart. The second stage will be to take the image from the camera directly beneath the Nerf gun, calculate the offset angle to the target, and swing the turret to the desired position. Once the turret is in place, the user will be free to adjust the elevation and fire the dart with the buttons. Once these two steps have been completed we will implement the fully automatic process.

The following functions form the bulk of the system and perform all the tasks in the process. The function center_target() is the first function called by start(). The first function called from center_target() is take_picture(). This function will interact with the camera driver to communicate over the UART with the camera and store the received jpeg. The jpeg is then passed into find_centroid() which uses a point of interest algorithm that returns the average position of the brightest points in the image. As long as there is only one distinct bright area on the image, this algorithm will point to the rough center of that area. Next center_target() calls set_motor() to change the azimuth of the turret so that the Nerf gun is pointing in the appropriate azimuth.

After the azimuth of the turret is adjusted start() calls range_target(). Similar to center_target(), range_target() will call take_picture() and pass the jpeg into find_centroid(); however, after the position of the first centroid is calculated the jpeg data will be purged from RAM to make room for another call to take_picture() and find_centroid() from the second camera. After the X position of the centroid in each of the pictures has been found, range_target will call calc_elevation(). calc_elevation() will call calc_range() to get a value for range which will be used as a key for a lookup table with data gathered experimentally for elevation angles to hit specific distances. The distance is calculated in calc_range() which uses centroid coordinates from both cameras and a triangulation formula to find the distance to the target. Finally range_target() will use set_motor() to adjust the elevation of the turret to the calculated angle.

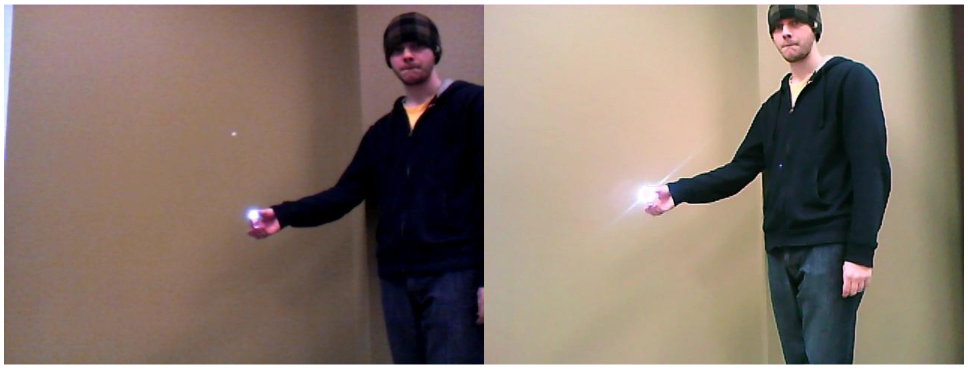
Once all the calculations have been completed, and the motors are in their desired positions, start() will call fire() which will call set_servo() to move the servo to push in the trigger. After the dart has been fired, start() will reset the motors to their initial states and return to the loop in start() to await another interrupt.

Test Plan

Our test plan is to utilize the modular hardware and software design of the system. It will be possible to test the Stepper Motor Control Circuit using the Oscilloscope, DC power supply and the wave generator that is available in the lab. Our software will be designed in an iterative manner so that as the hardware is interfaced to the micro-controller we will be able to write the relevant subroutines for the particular hardware. In this way we will be able to test out the stepper motor software controls by tying them to the four buttons on the micro-controller. Then we will be able to test the subroutines that will adjust the azimuth of the turret to center on the target in the picture; to do this requires only one camera to be interfaced. Last we will be able to add in the subroutines that will adjust the elevation of the turret. Each of the software subroutines will be written with a corresponding test subroutine. The test subroutine will fabricate input and verify the output of the code being tested.

Results of Experiments

After reading about the range finding formula we decided to try a dry run through. Both of us have Asus eeepcs with the same camera mounted on the screen, so we set them next to each other and took a single picture (at 640x489 resolution) of a bright white LED. Next we wrote a python script (see appendix for source code) that implements the point of interest search for the brightest spot in the jpeg described in the Software Design Section. Next the script takes the x location of the LED in each picture and uses the formula described in the Design and Description of Operation section to find the distance to the LED. The distance between the cameras was measured at 27.5cm and we physically measured the distance to the LED to be 1.91m. At first we used the value of P that we calculated for the SEN-09334 camera and the script returned a result of 3.2m as the distance to the LED. After searching for a more appropriate Field of View (75 degrees horizontal) for the camera that was used we tried the script with the value of P changed to 0.12 degrees per pixel. After this change the script returned 1.87m. Thus we are confident that we can calculate the distance to a target with reasonable accuracy given a picture from two cameras mounted parallel to each other.



Old Schedule

The following time estimates are based on the concept of a man week. With two people on the project and 7 weeks to work with we have 14 man weeks in which to allocate time. The following tasks then leave 2 man weeks as a fudge factor.

- Fabricate a turret platform with two axis of motion that can hold the Nerf gun mechanism and the firing servo
  - Approx. 0.5 Week
  - Group Member: Anonymous (Thanks to whoever built the turret in a past year)
  - Status: complete
- Wire Wrap Two Stepper Motor Control Circuits
  - Approx. 2 Weeks
  - Group Member: Unknown
- Fabricate a mount system to hold the two cameras with a parallel point of view.
  - Approx. 1 Week
  - Group Member: Unknown
- Interface the cameras with the micro-controller
  - Approx. 2 weeks
  - Group Member: Unknown
- Interface additional RAM with the micro-controller
  - Approx. 1 week
  - Group Member: Unknown
- Firing Subroutine:
  - Approx. 1 week
  - Group Member: Unknown
- Targeting Subroutine
  - Approx. 1 week
  - Group Member: Unknown
- Target Finding Subroutine
  - Approx. 1 week
  - Group Member: Unknown
- Range Finding Subroutine
  - Approx. 1 week
  - Group Member: Unknown
- Ballistic Model Subroutine
  - Approx. 1 week
  - Group Member: Unknown

New Schedule

- Fabricate a turret platform with two axis of motion that can hold the Nerf gun mechanism and the firing servo
  - Approx. 0.5 Week
  - Group Member: Anonymous (Thanks to whoever built the turret in a past year)
  - Status: complete
- Wire Wrap Two Stepper Motor Control Circuits
  - Approx. 2 Weeks
  - Group Member: Curtis
  - Status: complete and tested successfully
- Fabricate a mount system to hold the two cameras with a parallel point of view.
  - Approx. 1 Week
  - Group Member: Unknown
- Fabricate a mount for the firing servo
  - Approx. 1 Week
  - Group Member: Unknown
- Wire Wrap a breakout board for mounting on the turret so all of the signals are sent to the turret via a single ribbon cable.
  - Approx ½ Week:
  - Group Member: Curtis
- Interface the cameras with the micro-controller
  - Approx. 2 weeks
  - Group Member: Zach
  - Status: In Progress (~40%) – 2 way communication with the camera's work but not everything is supported by the device driver yet.
- Write a driver library for the Stepper Motor Controls
  - Approx ½ Week
  - Group Member: Curtis
  - Status: In Progress (~10%) – Clock pulse generation is close to working.
- Interface additional RAM with the micro-controller
  - Approx. 1 week
  - Group Member: Curtis
- Firing Subroutine:
  - Approx. 1 week
  - Group Member: Unknown
- Targeting Subroutine
  - Approx. 1 week
  - Group Member: Unknown
- Target Finding Subroutine
  - Approx. 1 week
  - Group Member: Unknown
- Range Finding Subroutine
  - Approx. 1 week

    - Group Member: Zach
- Ballistic Model Subroutine
    - Approx. 1 week
    - Group Member: Unknown
- Measure the System Power Usage for the datasheet
    - Approx ½ week
    - Group Member: Unknown

Citations

[1] Triangulation, Available: http://en.wikipedia.org/wiki/Triangulation, Last Modified: February 2nd, 2010, Last Accessed: February 4, 2010

[2] Hitec HS422 Data-sheet, Available: http://www.robotshop.ca/PDF/hs422.pdf

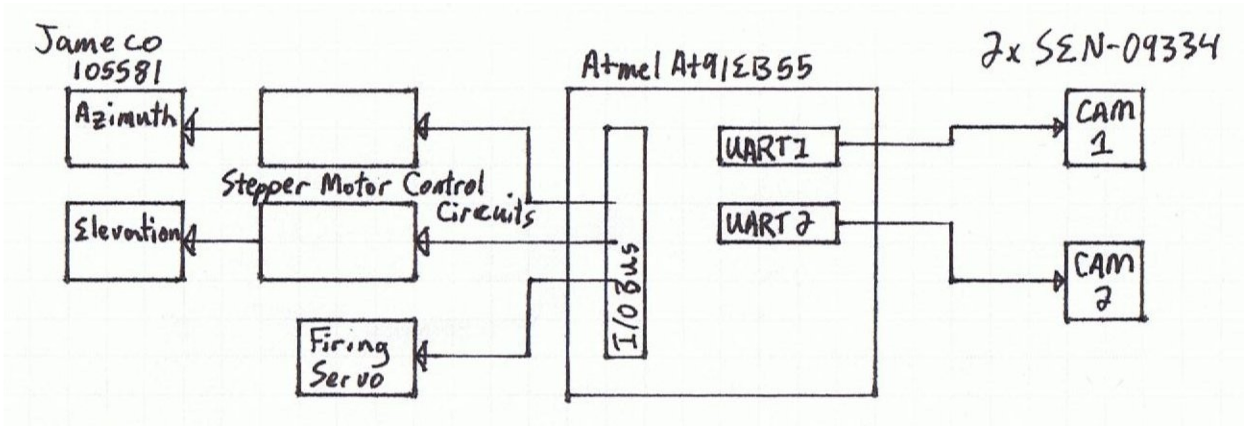
[3] Hitec HS635HB Data-sheet, Available: http://www.hitecrcd.com/product_file/file/56/HS635.pdf

[4] COMedia SEN-09334 JPEG UART Camera Data-sheet, Available: http://www.sparkfun.com/datasheets/Sensors/Imaging/C328.pdf

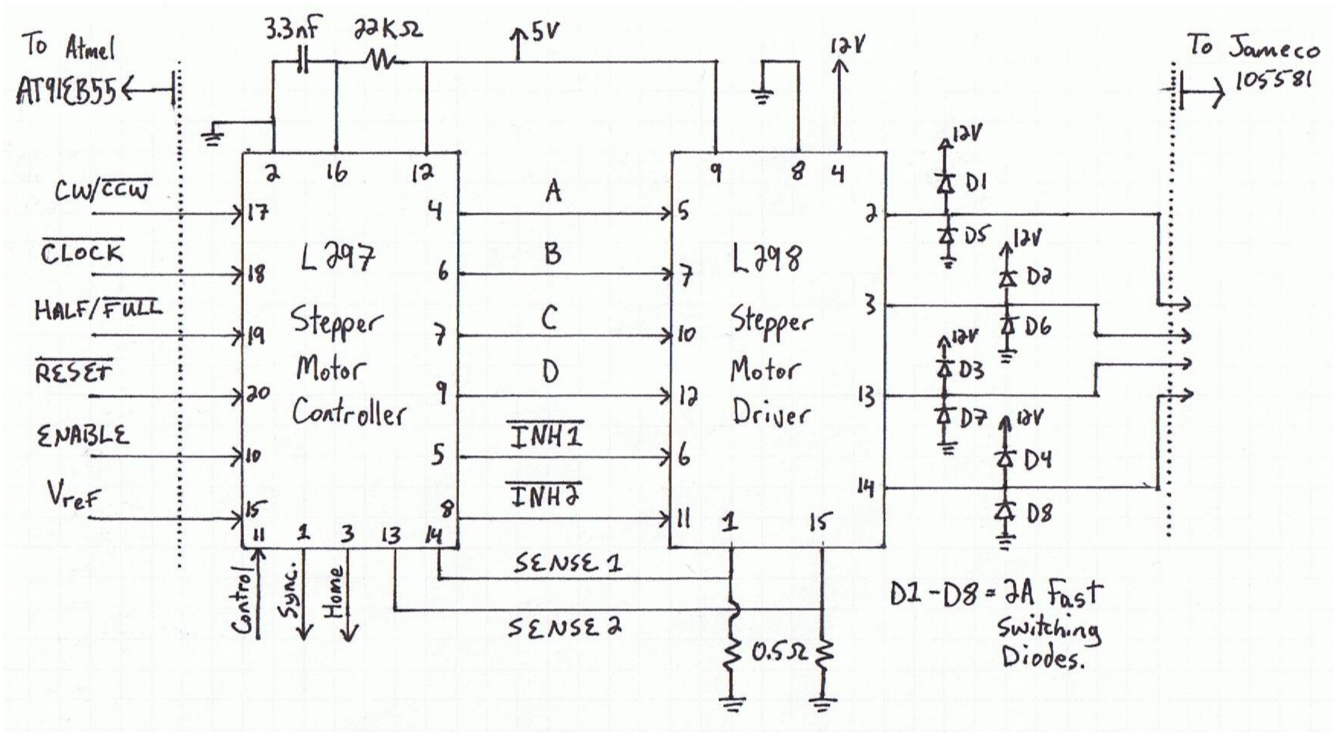
[5] L298 Stepper Motor Driver Data-sheet, Available: url=http://www.st.com/stonline/books/pdf/docs/1773.pdf

[6] L297 Stepper Motor Controller Data-sheet, Available: url=http://www.st.com/stonline/books/pdf/docs/1334.pdf

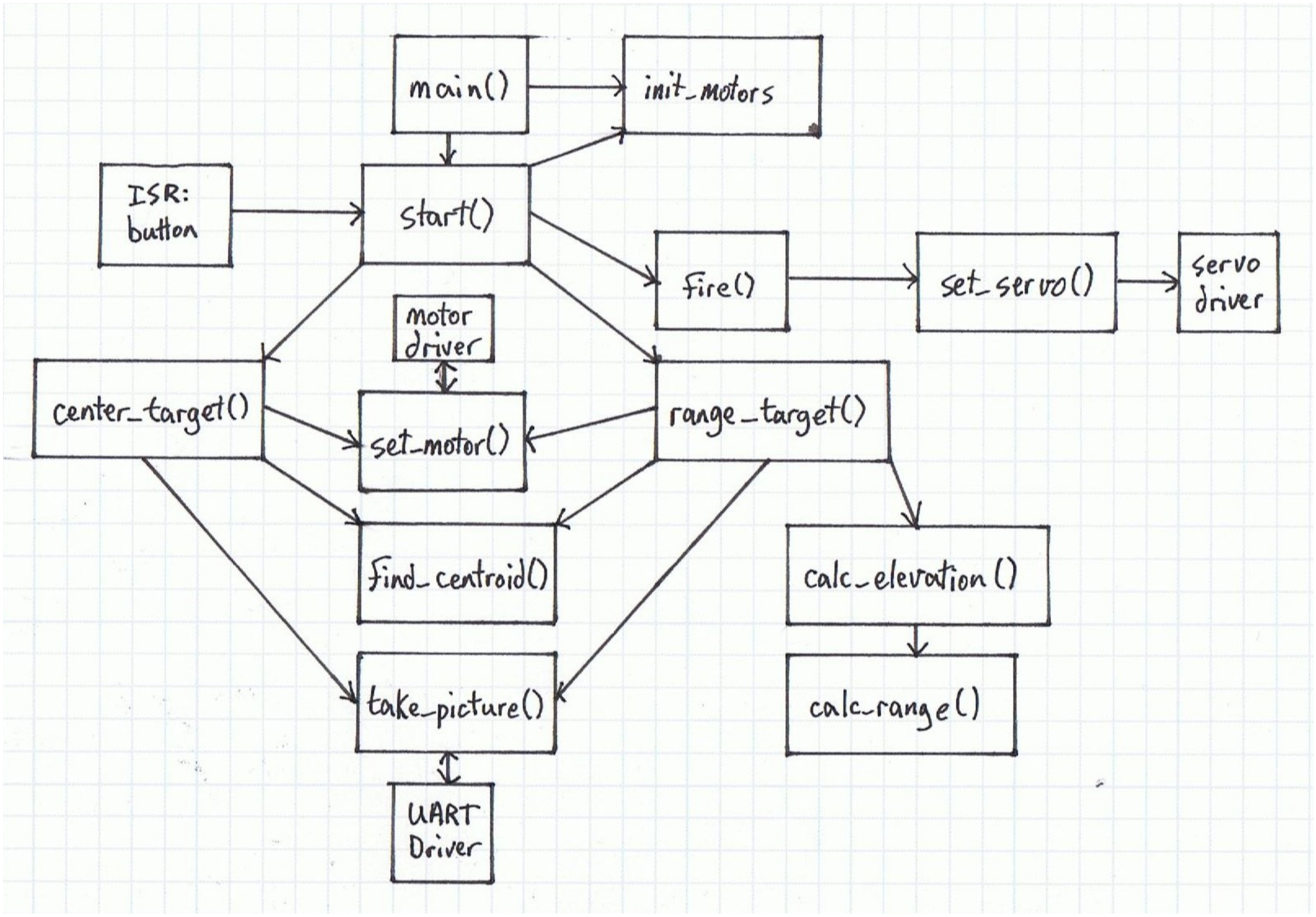
Hardware Block Diagram



Schematics



Graphical Hierarchy of Source Code



Source Code

Source Code Index

camera.h: Current version for the header of the camera driver. Not Compiled Successfully

camera.c: Current version of the camera driver. Not Compiled Successfully

spotfind.py: Test program for the range finding algorithm. Tested and Passed.

camera.h

```
/*Camera Driver Header file This file contains all the macros,
 * structure definitions, and function prototypes for the C328-7640
 * uart camera driver
 */

#ifndef CAMERA_H
```

```
#define CAMERA_H

#include "periph/usart/lib_usart.h"

/*Camera Instruction Set*/
/*1st Parameter  2nd Parameter  3rd Parameter  4th Parameter  */
#define INITIAL          0x01
/*0x00           colour type    preview res    jpeg resolution*/
#define GET_PICTURE      0x04
/*picture type   0x00           0x00           0x00           */
#define SNAPSHOT         0x05
/*snapshot type  skip byte 0    skip byte 1    0x00           */
#define SET_PACKAGE_SIZE 0x06
/*0x08           pkg size 0     pkg size 1     0x00           */
#define SET_BAUDRATE     0x07
/*first divider  second divider 0x00           0x00           */
#define RESET            0x08
/*reset type     0x00           0x00           0x00           */
#define POWER_OFF        0x09
/*0x00           0x00           0x00           0x00           */
#define DATA             0x0A
/*data type      data size 0    data size 1    data size 2    */
#define SYNC             0x0D
/*0x00           0x00           0x00           0x00           */
#define ACK              0x0E
/*command id     ack counter    pkg id 0       pkg id 1       */
#define NAK              0x0F
/*0x00           nak counter    error code     0x00           */
#define LIGHT_FREQUENCY  0x13
/*frequency type 0x00           0x00           0x00           */

/*Command Header and empty parameters*/
#define HEAD  0xAA
#define EMPTY 0x00
```

```
/*INITIAL Parameters*/
/*Color Type*/
#define COLOR_TYPE_2_BIT_GREY   0x01
#define COLOR_TYPE_4_BIT_GREY   0x02
#define COLOR_TYPE_8_BIT_GREY   0x03
#define COLOR_TYPE_12_BIT_COLOR 0x05
#define COLOR_TYPE_16_BIT_COLOR 0x06
#define COLOR_TYPE_JPEG         0x07

/*Preview Resolution*/
#define PREVIEW_RES_80x60   0x01
#define PREVIEW_RES_160x120 0x03
/*note: 0x05 and 0x07 will work here as well, although they are not
        described in the users manual explicitely and will give the
        same resolutions they do for JPEG resolution*/

/*JPEG Resolution*/
#define JPEG_RES_80x64   0x01
#define JPEG_RES_160x128 0x03
#define JPEG_RES_320x240 0x05
#define JPEG_RES_640x480 0x07

/*GET_PICTURE Parameters*/
/*Picture Type*/
#define PICTURE_TYPE_SNAPSHOT 0x01
#define PICTURE_TYPE_PREVIEW  0x02
#define PICTURE_TYPE_JPEG     0x05

/*SNAPSHOT Parameters*/
/*Sanpshot Type*/
#define SNAPSHOT_TYPE_COMPRESSED   0x00
#define SNAPSHOT_TYPE_UNCOMPRESSED 0x01

/*SET_BAUDRATE Parameters*/
/*1st Divider*/
```

```
#define BAUDRATE_7200   0xFF

#define BAUDRATE_9600   0xBF

#define BAUDRATE_14400  0x7F

#define BAUDRATE_19200  0x5F

#define BAUDRATE_28800  0x3F

#define BAUDRATE_38400  0x2F

#define BAUDRATE_57600  0x1F

#define BAUDRATE_115200 0x0F


/*2nd Divider*/

#define BAUDRATE_DIVIDER 0x01


/*RESET Parameters*/

/*Reset Type*/

#define RESET_HARD 0x00

#define RESET_SOFT 0x01


/*DATA Parameters*/

/*Data Type*/

#define DATA_TYPE_SNAPSHOT 0x01

#define DATA_TYPE_PREVIEW  0x02

#define DATA_TYPE_JPEG     0x05


/*NAK Parameters*/

/*Error Code*/

#define ERR_PIC_TYPE              0x01

#define ERR_PIC_UP_SCALE          0x02

#define ERR_PIC_SCALE             0x03

#define ERR_UNEXPECTED_REPLY      0x04

#define ERR_SEND_PIC_TIMEOUT      0x05

#define ERR_UNEXPECTED_CMD        0x06

#define ERR_SRAM_JPEG_TYPE        0x07

#define ERR_SRAM_JPEG_SIZE        0x08

#define ERR_PIC_FORMAT            0x09

#define ERR_PIC_SIZE              0x0A
```

```
#define ERR_PARAMETER             0x0B
#define ERR_SEND_REGISTER_TIMEOUT 0x0C
#define ERR_CMD_ID                0x0D
#define ERR_PIC_NOT_READY         0x0F
#define ERR_PACKAGE_NUMBER        0x10
#define ERR_WRONG_PACKAGE_SIZE    0x11
#define ERR_CMD_HEADER            0xF0
#define ERR_CMD_LENGTH            0xF1
#define ERR_SEND_PIC              0xF5
#define ERR_SEND_CMD              0xFF

/*Light Frequency Parameters*/
/*Frequency Type*/
#define FREQUENCY_50HZ 0x00
#define FREQUENCY_60HZ 0x01

/*Misc. Definitions*/
#define CMD_SIZE 6

/*Camera Descriptor Structure Definition*/
typedef struct
{
  UsartDesc * usart_desc; /*USART descriptor*/
  u_int       baud_rate;

} CameraDesc;

/*Command Frame Structure /definition*/
typedef struct
{
  char header;  /*always 0xAA*/
  char command; /*corresponds to the camera command*/
  char param1;  /*these correspond to parameters for the command*/
  char param2;
  char param3;
  char param4;
} CommandFrame;

/*Function Prototypes*/
u_int sync_camera ( CameraDesc * camera_desc );
u_int init_camera ( CameraDesc * camera_desc,
                    char colour_type,
                    char preview_res,
                    char jpeg_res );
void * take_picture ( CameraDesc * camera_desc,
                      char snapshot_type,
                      char picture_type,
                      u_int pkg_size );
CommandFrame * get_frame ( char * buffer );

#endif
```

camera.c

```
/*COMedia C328-7640 uart camera driver This file contains the function
 * definitions for the C328-7640 uart camera
 */
#include <stdlib.h>
#include <string.h>

#include "camera.h"
#include "parts/m55800/lib_m55800.h"
#include "drivers/capture/capture.h"
#include "drivers/wait/wait.h"
#include "parts/m55800/eb55.h"

extern void wake_up_handler (void) ;

WaitDesc wait_desc = { &TC0_DESC, 0, 0, WAIT_DELAY, wake_up_handler } ;
```

```
/* Synchronize the camera for use Opens the usart, then goes through
 * the SYNC->, <-ACK SYNC, ACK-> procedure with the camera Returns
 * true on successful synchronization, false otherwise.
 */
u_int sync_camera ( CameraDesc * camera_desc )
{
  u_int i, status, return_val = FALSE;
  char * buffer;
  CommandFrame sync = { HEAD, SYNC, EMPTY, EMPTY, EMPTY, EMPTY };
  CommandFrame ack  = { HEAD, ACK , SYNC , EMPTY, EMPTY, EPMTY };
  CommandFrame rec_ack, rec_sync;
  /*Set clock for wait function*/
  wait_desc.mcki_khz = 32000; /*clock speed: 32kHz*/
  wait_desc.period = 1000; /*time between transmissions: 1s*/

  /*Create the buffer*/
  buffer = malloc ( 2 * CMD_SIZE );
  memset ( buffer, 0, 2 * CMD_SIZE );

  /*Open the usart*/
  at91_usart_open ( camera_desc->usart_desc,
                    US_ASYNC_MODE,
                    camera_desc->baud_rate, 0 );

  /*Set up the receive buffer*/
  at91_usart_receive_frame ( camera_desc->usart_desc,
                             buffer,
                             2 * CMD_SIZE,
                             0 );

  /*Send the sync command*/

  /*Camera needs sync sent up to 60 times to detect baudrate*/
  for ( i = 0 ; i < 60 ; i++ )
    {
      /*Send SYNC*/
      at91_usart_send_frame ( camera_desc->usart_desc,
                              (char*) sync, CMD_SIZE );
      /*Delay before next attempt*/
      at91_wait_open ( & wait_desc );
      status = at91_usart_get_status ( camera_desc->usart_desc );

      /*Check if something was received, need to refactor this section*/
      /* using get_frame()*/
      if ( status & US_RXRDY )
      {
          /*Get first received frame*/
        rec_ack = (CommandFrame) buffer;
          /*Get second received frame*/
        rec_sync = (CommandFrame) ( buffer + CMD_SIZE );

        if ( rec_ack.command ==
               ACK && rec_ack.param1 ==
               SYNC && rec_sync.command ==
               SYNC )
          {
              /*Send ACK*/
            at91_usart_send_Frame ( camer_desc->usart_desc,
                                      (char*) ack, CMD_SIZE );
            return_val = TRUE;
          }
        break;
      }

    }
  free ( buffer );
  return return_val;
}

/*Initializes the camera for use Builds the frame to be used by the camera,
 * sends the INITIAL command to the camera, and receives the ACK from the
 * camera.  Returns true on successful initialization, false otherwise
 */
u_int init_camera ( CameraDesc * camera_desc,
                    char colour_type,
                    char preview_res,
                    char jpeg_res )
{}

/*Gets a picture from the camera Sends all the commands required to get a
 * picture from the camera.  Sends SET_PACKAGE_SIZE, then SNAPSHOT, then
 * GET_PICTURE.  Camera sends DATA command containing the size of the image to
 * be sent to the board.  The image is then sent in packets and the image data
 * is extraced and stored in a buffer.  That buffer is returned
 */
void * take_picture ( CameraDesc * camera_desc,
                      char snapshot_type,
                      char picture_type,
                      u_int pkg_size )
{}

/*Finds a valid frame in a buffer Looks for a 0xAA in the buffer and returns
 * that point as a CommandFrame pointer
 */
CommandFrame * get_frame ( char * buffer )
{}
```

spotfind.py

```
#!/usr/bin/python

#test of range finding, the two provided jpgs look at a light source
#1.9m away

import math, os, sys
```

```
from PIL import Image

#findBlob takes an array of RGB pixels and returns the
#centroid of the bright spot
def findBlob(pix):
    numPix = 0
    xMag = 0
    highPass = 750

    for x in range(640):
        for y in range(480):
            pixel = pix[x,y]
            light = pixel[0]+pixel[1]+pixel[2]
            if light >= highPass:
                xMag = xMag + x
                numPix = numPix + 1
    xCentre = xMag/numPix

    return xCentre

#findRange takes two x coordinates and uses simple
#triangulation to find the distance to the light source
def findRange(x1, x2):
    camWidth = 0.275  #distance between cameras
    degPerPix = 0.12  #degrees per pixel on the image

    dev1 = getDev(x1)
    dev2 = getDev(x2)

    alpha = math.radians(90 - dev1 * degPerPix)
    beta = math.radians(90 - dev2 * degPerPix)

    range = camWidth / ((1 / math.tan(alpha)) + (1 / math.tan(beta)))
    return range
```

```
#getDev is a helper that correctly finds the deviation
#from the centre of the image
def getDev(x):
    if x > 320:
        dev = x - 320
    else:
        dev = 320 - x
    return dev



if __name__ == '__main__':
    im1 = Image.open("test1.jpg")
    im2 = Image.open("test2.jpg")
    pix1 = im1.load()
    pix2 = im2.load()
    print findRange(findBlob(pix1), findBlob(pix2))
```