

Final Report

Olugbenga Moses Anubi

Robot Name: PingBot

EEL 5666 Intelligent Machines Design Laboratory

Instructor Names:

- Dr. Arroyo
- Dr. Schwartz

TA Names:

- Thomas Vermeer
- Mike Pridgen

TABLE OF CONTENTS

Abstract.....	5
Introduction	5
Background Information that leads into the problem	5
Objective.....	5
Course Objectives	5
My Objective.....	5
Robot’s Objective.....	6
Scope.....	6
Integrated System.....	6
Mobility.....	7
Hardware	7
Software.....	7
Obstacle Avoidance	7
Hardware	7
Software.....	8
Vision.....	8
Hardware	8
Software.....	8
Manipulation.....	8
Hardware	8
Software.....	8
Display.....	8
Hardware	8

Software.....	9
Gripping.....	9
MECHANICAL PLATFORM.....	9
appendix.....	12
Details of CMU CAM	12
Parts That Ships with CMUcam 1.....	12
Power	12
Level Shifted Serial Port	12
TTL Serial Port	12
Programming Port.....	12
Camera Bus	12
Servo Port.....	13
Jumpers.....	13
Parallel Processing in Slave Mode (Jumper 1)	13
Baud Rate (Jumpers 2 and 3)	13
Demo Mode (Jumper 4).....	13
Serial Command Set.....	14
Output Data Packet Descriptions.....	25
SOFTWARE	27
main body of the software	27
PVR.c (from mike and thomas, teaching assistants).....	50
uart.c (originally from hao he but modified by me to work with PINGBOT).....	59
references	69

ABSTRACT

The name of the Robot described in this report is PingBot. Its objective is to locate, collect and deliver ping pong balls. It locates the balls using a special sensor, the CMU CAM 1. It avoids obstacle using fuzzy logic synthesized from the real-time information gotten from Analog Infrared Sensors. The picking mechanism for the robot is designed around a vacuum pump. The design schematic in form of a block diagram for the integrated system is also presented. The objective and functions of the various hardware and related software subroutines are also described.

INTRODUCTION

Robot's name: PingBot.

BACKGROUND INFORMATION THAT LEADS INTO THE PROBLEM

I am a student in the IMDL class which is recommended for all the students in CIMAR. I want to design and build a robotic platform and program it to perform the task I want, which is to locate, identify, pick and deliver ping pong balls.

OBJECTIVE

COURSE OBJECTIVES

The Intelligent Machines Design Laboratory (IMDL) constitutes a capstone undergraduate laboratory and a beginning graduate laboratory that provides students with a realistic engineering experience in design, simulation, fabrication, assembly, integration, testing, and operation of a relatively complex, intelligent machine. A course project, oriented about a small, microcomputer controlled, electronically sensualized, autonomous mobile robot that exhibits various tasking behaviors, requires the integration of various sub-disciplines in electrical and computer engineering: microcomputer interfacing and programming, analog and digital electronics, computer-aided engineering, control, and communications.

MY OBJECTIVE

Design, build, and program an autonomous mobile robot using kit parts combined with novel circuits and mechanics of my own design to meet the requirements and objective of the course.

ROBOT'S OBJECTIVE

Locate, identify, pick and deliver ping pong balls

SCOPE

Designed to meet the course requirements

Process the following sensors

- IR proximity detectors
- Bump switches
- Vision sensors (CMU camera)
- Tactile sensors

INTEGRATED SYSTEM

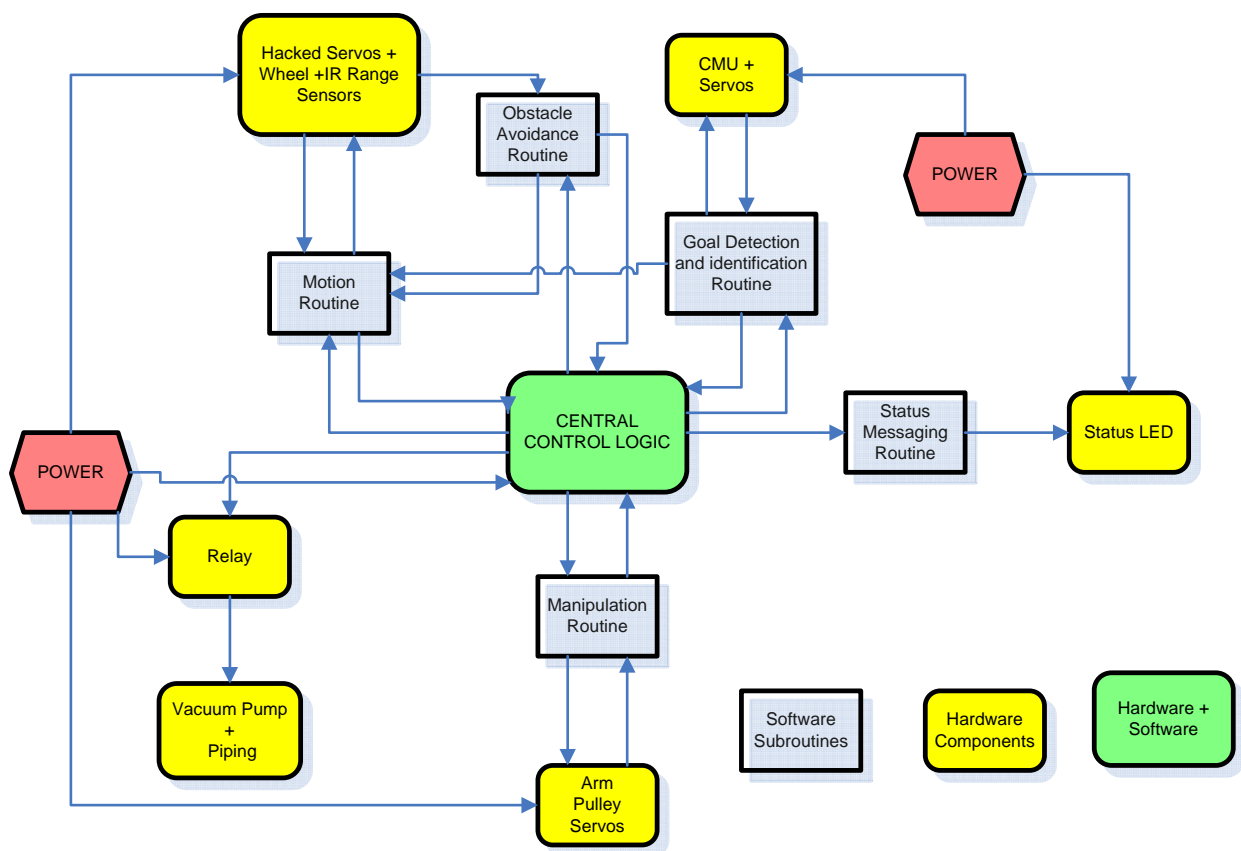


Fig. 1 Block diagram for the integrated system

As shown in the block diagram of Fig 1, the functionality of the robot is broken down to the following functional headings:

- Mobility
- Obstacle Avoidance
- Vision
- Manipulation
- Display
- Gripping

While some of these headings are purely software subroutines, some of them are hardware compare components, and some comprise both.

MOBILITY

HARDWARE

Mobile Platform with electric motors (DC or Hacked Servos) and wheels for moving the robot around.

SOFTWARE

The software component for this function would implement a set of subroutines for driving straight, turning in a given direction and carrying out other necessary POSE (Position and Orientation) related tasks.

OBSTACLE AVOIDANCE

HARDWARE

Three units of IR range sensors systematically arranged to ensure detection of obstacles in the robot's path. This is done using fuzzy logic synthesized from the information obtain from the IR sensors. Basically, the three IRs were used and mounted left, center, and right. The left and right IRs determine the drive direction of the robot. The center one helps prevent bumping into obstacles not picked up by the other two IRs.

SOFTWARE

Implements routines that interpret the signals from the IR range sensors and interact with the vision algorithms to differentiate obstacles from target objects (goal) and send require POSE correction signals to the Mobility algorithm so that the robot can effectively avoid obstacle in its path while still oriented towards the target. The codes for this is attached.

VISION

HARDWARE

The hardware setup, consisting of a CMU camera and two mini-servos, keeps track of target objects. In the appendix is the detailed information on the CMU CAM used in the design.

SOFTWARE

Implements subroutines that get target coordinates from the hardware setup and also assists the obstacle avoidance subroutine to different target from obstacles.

MANIPULATION

HARDWARE

Mechanical Arm plus pulleys, Servos, two Limit Switches (This counts for bump sensors), tactile sensors. The tactile sensors will be mounted on the tip of the gripper so as to know when the object is picked (sucked by the vacuum)

SOFTWARE

Routine ensures accurately positioning of the arm depending on the nature of the signals received from the vision system

DISPLAY

HARDWARE

LCD for displaying messages of what the robot is thinking at the moment

SOFTWARE

The software routines handles the creation of the necessary messages of what is going on with the robot at the moment for debugging purposes as well as demo and presentation purposes

GRIPPING

The gripping system is purely a hardware system that uses vacuum to suck up the target object. The intended vacuum pump requires a 7.2V power source. The logical switching of this device would be accomplished using a relay.

MECHANICAL PLATFORM

The figure below shows the main idea behind the platform of the proposed robot. The Arm assembly is a two-degree-of-freedom device whose control is decoupled from the main control of the mobile platform itself. The platform drives straight in forward and reverse direction as well turn in place using differential wheel system and a roller caster for stability. The actuation for the platform is ML-30 DC motors from Robotmarketplace.

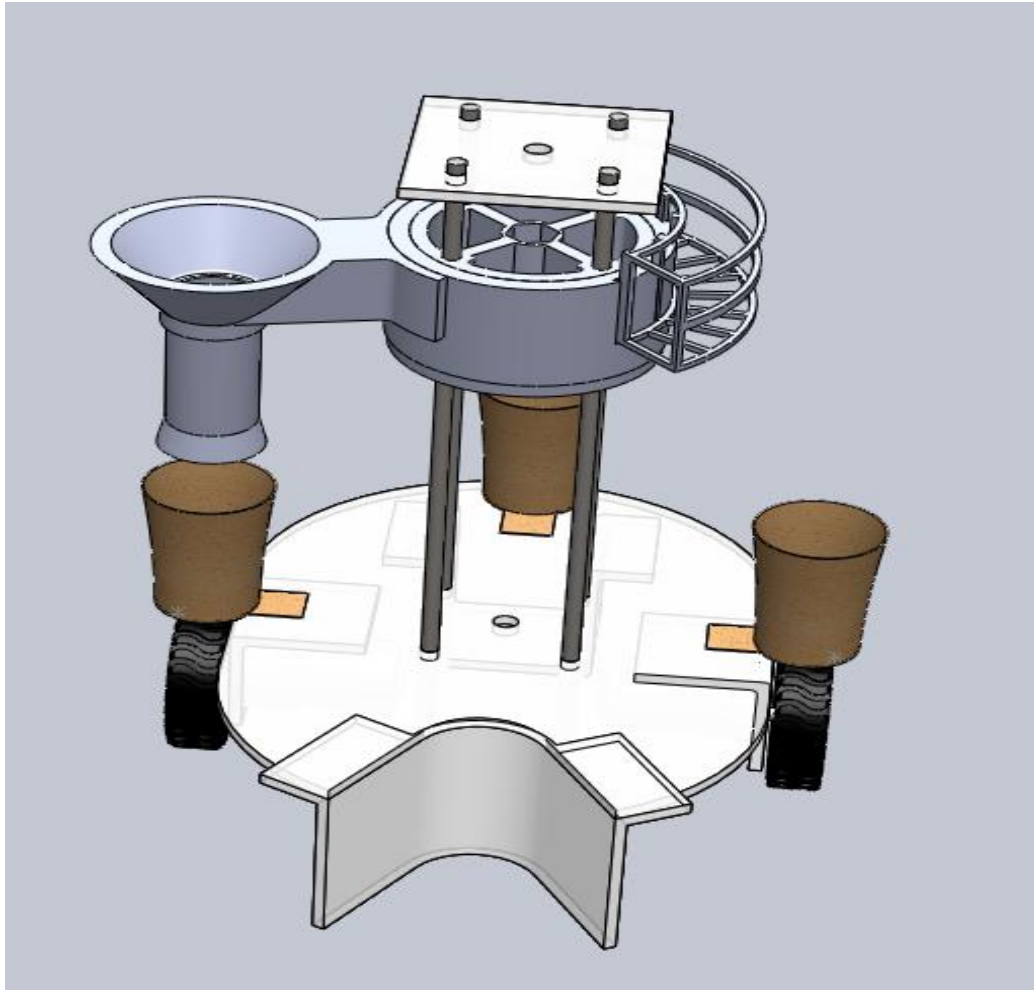
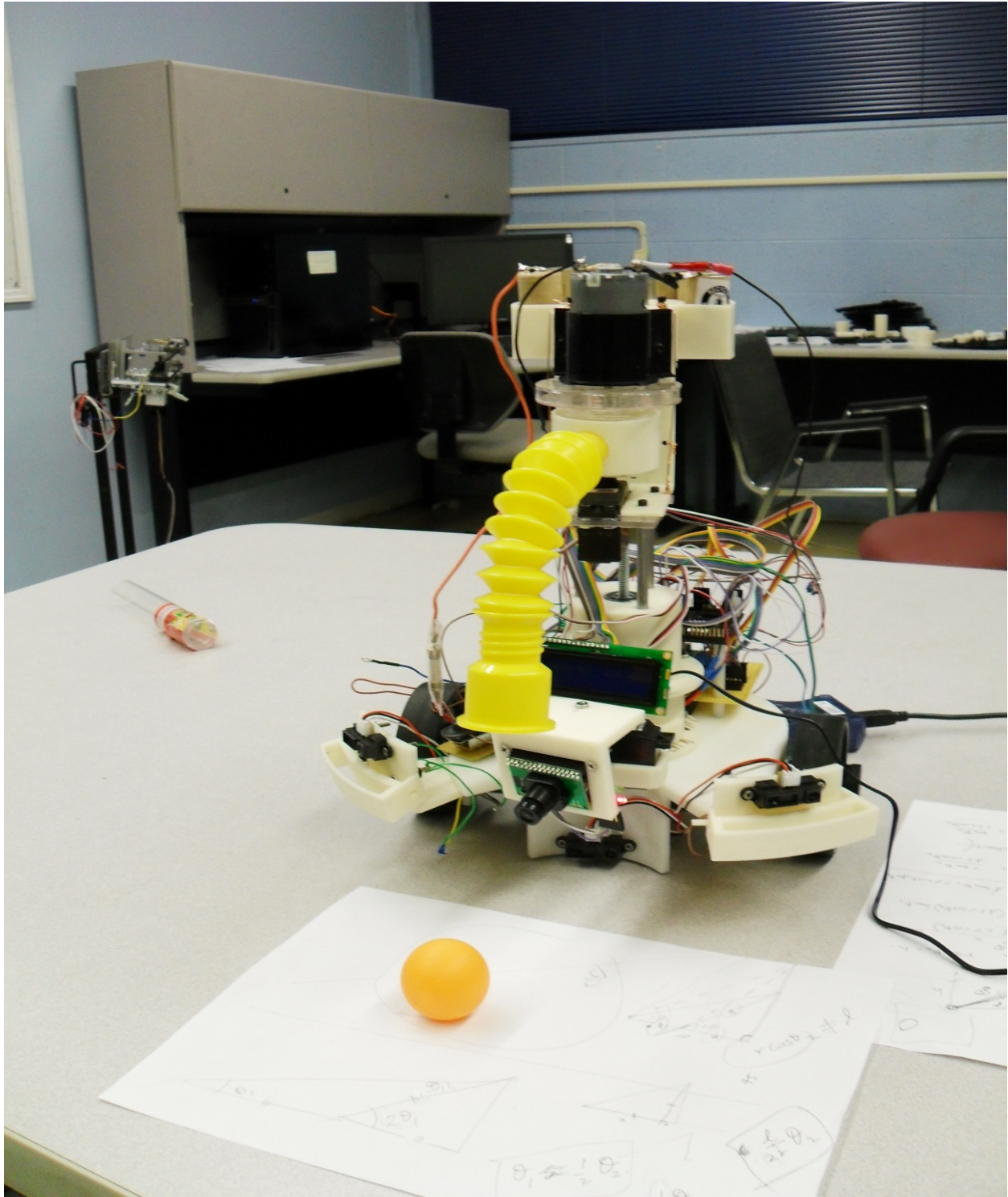


Figure. Solidworks Model of the Robot.



Picture of the Actual Robot

APPENDIX

DETAILS OF CMU CAM

PARTS THAT SHIPS WITH CMUCAM 1

The camera consists of two major components, the CMUcam board and the CMOS camera module. This CMOS camera module must be attached to the CMUcam at all times in order for the system to function. From power up, the camera can take up to 15 seconds to automatically adjust to the light. Pin 32 on the camera bus is a black and white analog output pin. It is possible to connect the analog output to a TV or multi-sync monitor. Due to the clock rate of the camera, the analog output does not correctly synchronize with standard NTSC or PAL systems. Other parts include:

POWER

The input power to the board goes through a 5 volt regulator. The ideal power supply to the board is between 6 and 9 volts of DC power that is capable of supplying at least 200 milliamperes of current.

LEVEL SHIFTED SERIAL PORT

This port provides full level shifting for communication with a computer. Though it only uses 3 of the 10 pins it is packaged in a 2x5 pin configuration to fit standard 9 pin ribbon cable clip-on serial sockets and 10 pin female clip on serial headers that can both attach to a 10 wire ribbon cable.

TTL SERIAL PORT

This serial port taps into the serial I/O before it goes through the MAX232 chip. This port may be ideal for communication with a microcontroller that does not have any built-in level shifting.

PROGRAMMING PORT

The programming port allows the firmware to be downloaded to the SX28 using a SX-Key / Blitzer or equivalent programmer.

CAMERA BUS

This bus interfaces with the CMOS camera chip. The CMOS camera board is mounted parallel to the processing part of the board and connects starting at Pin 1

SERVO PORT

This is the output for the servo. The servo power does not go through a regulator from the board's power input

JUMPERS

PARALLEL PROCESSING IN SLAVE MODE (JUMPER 1)

The CMUcam supports a mode of operation that allows multiple boards to process data from the same camera. If a PC104 style pass-through header is used instead of the standard double row female header, it is possible to rack multiple boards along the same camera bus. Upon startup, if jumper 1 is set, the camera becomes a slave. Slave mode stops the camera board from being able to configure or interfere with the CMOS camera's settings. Instead it just processes the format setup by the master vision board. When linking the buses together the user must only have one master; all other boards should be setup to be in slave mode.

BAUD RATE (JUMPERS 2 AND 3)

115,200 Baud	Jumper 2 Open	Jumper 3 Open
38,400 Baud	Jumper 2 Set	Jumper 3 Open
19,200 Baud	Jumper 2 Open	Jumper 3 Set
9,600 Baud	Jumper 2 Set	Jumper 3 Set

Because of the extra time it takes to transmit data at lower rates, frames may be skipped, resulting in a lower frame rate. The slower rate will also cause the bitmap mode and dump frame resolutions to shrink.

DEMO MODE (JUMPER 4)

Jumper 4 puts the camera into a demo mode. Demo mode causes the camera to call the track window command and then begin outputting a standard hobby servo PWM signal from the servo output. The servo attempts to drive the camera mounted on it towards the middle mass of the color detected on startup.

SERIAL COMMAND SET

The serial communication parameters are as follows:

- 115,200 Baud
- 8 Data bits
- 1 Stop bit
- No Parity
- No Flow Control (Not Xon/Xoff or Hardware)

All commands are sent using visible ASCII characters (123 is 3 bytes "123"). Upon a successful transmission of a command, the ACK string should be returned by the system. If there was a problem in the syntax of the transmission, or if a detectable transfer error occurred, a NCK string is returned. After either an ACK or a NCK, a `\r` is returned. When a prompt (`\r` followed by a `:`) is returned, it means that the camera is waiting for another command in the idle state. White spaces do matter and are used to separate argument parameters. The `\r` (ASCII 13 carriage return) is used to end each line and activate each command. If visible character transmission exerts too much overhead, it is possible to use varying degrees of raw data transfer.

`\r`

This command is used to set the camera board into an idle state. Like all other commands, the user should receive the acknowledgment string "ACK" or the not acknowledge string "NCK" on failure. After acknowledging the idle command the camera board waits for further commands, which is shown by the `:` prompt. While in this idle state a `/r` by itself will return an "ACK" followed by `\r` and `:` character prompt.

Example of how to check if the camera is alive while in the idle state

`:`

ACK

`:`

CR [reg1 value1 [reg2 value2 ... reg16 value16]]`\r`

This command sets the Camera's internal **R**egister values directly. The register locations and possible settings can be found in the Omnivision CMOS camera documentation. All the data sent to this command should be in decimal visible character form unless the camera has previously been set into raw mode. It is possible to send up to 16 register-value combinations. Previous register settings are not reset between CR calls; however, the user may overwrite previous settings. Calling this command with no arguments resets the camera and restores the camera registers to their default state. This command can be used to hard code gain values or manipulate other low level image properties.

Common Settings:

<u>Register</u>	<u>Values</u>	<u>Effect</u>
5 Contrast	0-255	
6 Brightness	0-255	
18 Color Mode		
36		YCrCb* Auto White Balance On
32		YCrCb* Auto White Balance Off
44		RGB Auto White Balance On
40		RGB Auto White Balance Off (default)
17 Clock Speed		
	2	17 fps (default)
	3	13 fps
	4	11 fps
	5	9 fps
	6	8 fps
	7	7 fps
	8	6 fps
	10	5 fps
	12	4 fps
19 Auto Exposure		
	32	Auto Gain Off
	33	Auto Gain On (default)

Example of decreasing the internal camera clock speed (default speed is 2)

:CR 17 5

ACK

:

*The red channel becomes Cr which approximates r-g, The green channel becomes Y which approximates intensity, the blue channel becomes Cb which approximates b-g

RGB -> CrYCb

$Y=0.59G + 0.31R + 0.11B$

$Cr=R-Y$

$Cb=B-Y$

DF\r

This command will **D**ump a **F**rame out the serial port to a computer. This is the only command that by default only returns a non-visible ASCII character packet. It dumps a type F packet that consists of the raw video data column by column with a frame synchronize byte and a column synchronize byte. (This data can be read and displayed by the CMUcamGUI java application.) Since the data rate required to send the raw video greatly exceeds the maximum serial port speed, only one column per frame is sent at a time

Type F data packet format

1 - new frame

2 - new col

3 - end of frame

RGB (CrYCb) ranges from 16 - 240

1 2 r g b r g b ... r g b r g b 2 r g b r g b r ... r g b r g b ...

Example of a Frame Dump from a terminal program :

(WARNING: This may temporarily interfere with a terminal program by sending non-visible characters)

:DF

ACK

*maKP(U A\$IU AL<>U A\$L*YL%*L L (G AUsonthAYA(KMAy098a34ymawvk...*

DM value\r

This command sets the **D**elay before characters that are transmitted over the serial port and can give slower processors the time they need to handle serial data. The value should be set between 0 and 255. A value of 0 (default) has no delay and 255 sets the maximum delay. Each delay unit correlates to approximately the transfer time of one bit at the current baud rate.

GM\r

This command will **G**et the **M**ean color value in the current image. If, optionally, a subregion of the image is selected, this function will only operate on the selected region. The mean values will be between 16 and 240 due to the limits of each color channel on the CMOS camera (See page 10). It will also return a measure of the average absolute deviation of color found in that region. The mean together with the deviation can be a useful tool for automated tracking or detecting change in a scene. In YCrCb mode RGB maps to CrYCb.

Type S data packet format

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

Example of how to grab the mean color of the entire window

:SW 1 1 40 143

ACK

:GM

ACK

S 89 90 67 5 6 3

S 89 91 67 5 6 2

GV\r

This command **G**ets the current **V**ersion of the firmware from the camera. It returns an ACK followed by the firmware version string.

Example of how to ask for the firmware version:

:GV

ACK

CMUcam v1.12

HM active\r

This command puts the camera into **H**alf-horizontal resolution **M**ode for the DF command and the LM command when dumping a bitmap image. An *active* value of 1 causes only every odd column to be processed. The default value of 0 disables the mode.

II \r

This command uses the servo port as a digital Input. Calling I1 returns either a 1 or 0 depending on the current voltage level of the servo line. The line is pulled high; because of this it is only required to pull it low or let it float to change its state. The servo line can also be used as a digital output.

Example of how to read the digital value of the servo line:

:I1

ACK

1

L1 *value\r*

This command is used to control the tracking Light. It accepts 0, 1 and 2 (default) as inputs. 0 disables the tracking light while a value of 1 turns on the tracking light. A value of 2 puts the light into its default auto mode. In auto mode, and while tracking, the light turns on when it detects the presence of an object that falls within the current tracking threshold. This command is useful as a debugging tool.

Example of how to toggle the Tracking Light on and then off:

:L1 2

ACK

:L1 0

ACK

LM *active\r*

This command turns on Line Mode which uses the time between each frame to transmit more detailed data about the image. It adds prefix data onto either **C**, **M** or **S** packets. This mode is intended for users who wish to do more complex image processing on less reduced data. Since the frame rate is not compromised, the actual processing of the data put out by the vision system must be done at a higher rate. This may not be suitable for many slower microcontrollers.

Line mode's effect on TC and TW:

When line mode is active and TC or TW is called, line mode will send a binary bitmap of the image as it is being processed. It will start this bitmap with a 0xAA flag value (hex value AA not in human readable form). The value

0xAA will not occur in the data stream. This is followed by bytes each of which contains the binary value of 8 pixels being streamed from the top-left to the bottom-right of the image. The vertical resolution is constrained by the transfer time of the horizontal data so lines may be skipped when outputting data. In full resolution mode, the resulting binary image is 80x48. The binary bitmap is terminated by two 0xAA's. This is then followed by the normally expected standard **C** or **M** data packet (processed at that lower resolution).

Example of TC with line mode on:

:LM 1

:TC

(raw data: AA XX XX XX XX XX XX AA AA) C 45 72 65 106 18 51

(raw data: AA XX XX XX XX XX XX AA AA) C 46 72 65 106 18 52

Line mode's effect on GM:

When line mode is active and GM is called, line mode will send a raw (not human readable) mean value of every line being processed. These packets are started with a 0xFE and terminate with a 0xFD. Each byte of data between these values represents the corresponding line's mean color value. Similarly to the bitmap mode the vertical resolution is halved, because of the serial transfer time. At 17 fps 115,200 baud every other line is skipped. At any slower frame rate, (still 115,200 baud) no lines will be skipped.

Example of GM with line mode on:

:LM 1

:GM

(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8

(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8

MM *mode*\r

This command controls the **M**iddle **M**ass mode which adds the centroid coordinates to the normal tracking data. A *mode* value of 0 disengages middle mass, a value of 1(default) engages middle mass and a value of 2 engages the mode and turns on the servo PWM signal that tries to center the camera on the center of color mass (see the Demo Mode Jumper description). The *mode* value acts as a bit field for the following operations. Setting the bits enables the functionality.

mode = B₃ B₂ B₁ B₀

Bits

B₀ Enable / Disable Middle Mass Mode

B₁ Enable / Disable the Servo Output

B₂ Change Servo Direction

B₃ Return N-type packet that includes the current servo position (see page 27.)

Example of how to disable Middle Mass mode:

:MM 0

ACK

:TC

ACK

C 38 82 53 128 35 98

C 38 82 53 128 35 98

C 38 82 53 128 35 98

NF *active*\r

This command controls the **Noise Filter** setting. It accepts a Boolean value 1 (default) or 0. A value of 1 engages the mode while a value of 0 deactivates it. When the mode is active, the camera is more conservative about how it selects tracked pixels, it requires 2 sequential pixels for a pixel to be tracked.

Example of how to turn off noise filtering:

:NF 0

ACK

:

PM *mode*\r

This command puts the board into **Poll Mode**. Setting the mode parameter to 1 engages poll mode while 0 (default) turns it off. When poll mode is engaged only one packet is returned when an image processing function is called.

Example of how to get one packet at a time:

:PM 1

ACK

:TC 50 20 90 130 70 255

ACK

C 38 82 53 128 35 98

:

RM *bit_flags*\r

This command is used to engage the **Raw** serial transfer **Mode**. It reads the bit values of the first 3 (lsb) bits to configure settings. All bits cleared sets the default visible ascii mode. If bit 0 is set, then all output from the camera is in raw byte packets. The format of the data packets will be changed so as not to include spaces or be formatted as readable ASCII text. Instead the user will receive a 255 valued byte at the beginning of each packet, the packet identifying character (i.e. C for a color packet) and finally the packet. There is no \r sent after each packet, so the user must use the 255 to synchronize the incoming data. Any 255 valued bytes that may be sent as part of the packet are set to 254 to avoid confusion. If bit 1 is set, the “ACK\r” and “NCK\r” confirmations are not sent. If bit 3 is set, input will be read as raw byte values, too. In this mode, after the two command byte values are sent, send 1 byte telling how many arguments are to follow. (i.e. DF followed by the raw byte value 0 for no arguments) No \r character is required.

bit_flags = B₂ B₁ B₀

Bits

B₀ Output to the camera is in raw bytes

B₁ “ACK\r” and “NCK\r” confirmations are suppressed

B₂ Input to the camera is in raw bytes

Example of the new packet for Track Color with Raw Mode output only :

(WARNING: This may temporarily interfere with a terminal program by sending non visible characters)

:RM 1

ACK

:TC 50 20 90 130 70 255

ACK

C>%k(ai Ck\$&,.L

RS \r

This command **ReSets** the vision board. Note, on reset the first character is a /r.

Example of how to reset the camera:

:rs

ACK

CMUcam v1.12

:

S1 *position* \r

This command lets the user **Set** the position of servo 1. 0 turns the servo off and holds the line low. 1-127 will set the servo to that position while it is tracking or getting mean data. Any value 128 and higher sets the line high. In order for the servo to work, the camera must be in either a tracking loop or mean data gather loop. Values 0 and 128 can be useful if the user wish to use the servo port as a digital output. The port can also be used as a digital input (see I1 command). The “MM” command can enable or disable the automatic servo tracking.

SM *value* \r

This command is used to enable the **Switching Mode** of color tracking. When given a 0 it is in its default mode where tracking will return its normal C or M color packet. If the value is set to 1, the tracking commands will alternate each frame between color packets and S statistic packets. Each statistic packet is only being sampled from an area one quarter the size of the bounded area returned from the tracking command. If no object was bounded, then no S statistic packets are returned. This can be useful for adaptive tracking or any type of tracking where the user would like to get feedback from the currently bound target. After the first tracking packet is returned, the window gets set back to full size for all future packets

Example of how to Track Color with SM:

:SM 1

ACK

:TC 200 255 0 30 0 30

ACK

C 2 40 12 60 10 70

S 225 20 16 2 3 1

C 5 60 20 30 12 100

S 225 19 17 1 2 1

C 0 0 0 0 0 0

C 0 0 0 0 0 0

C 0 0 0 0 0 0

C 5 60 20 30 12 100

S 225 19 17 1 2 1

SW [x y x2 y2] \r

This command **Sets the Window** size of the camera. It accepts the x and y Cartesian coordinates of the upper left corner followed by the lower right of the window the user wish to set. The origin is located at the upper left of the field of view. **SW** can be called before an image processing command to constrain the field of view. Without arguments it returns to the default full window size of 1,1,80,143.

Example of setting the camera to select a mid portion of the view:

:SW 35 65 45 75

ACK

:

TC [Rmin Rmax Gmin Gmax Bmin Bmax] \r

This command begins to **Track a Color**. It takes in the minimum and maximum RGB (CrYCb) values and outputs a type M or C or N data packet (set by the MM command). The smaller type C packet encodes a bounded box that contains pixels of the currently defined color, the number of found pixels scaled (actual value is (pixels+4)/8) that fall in the given color bounds and a confidence ratio. The default type M packet also includes the center of mass of the object. The resolution of the processed image is 80x143. The X values will range from 1 to 80 and the y values will range from 1 to 143. A packet of all zeros indicates that no color in that range was detected. The confidence value is a ratio of the pixels counted within the given range and the area of the color bounding box, it returns a value which ranges from 0 to 255. Under normal operations any value greater than 50 should be considered a very

confident lock on a single object. A value of 8 or lower should be considered quite poor. With no arguments, the last color tracking parameters will be repeated.

Type M packet

M mx my x1 y1 x2 y2 pixels confidence\r

Type C packet

C x1 y1 x2 y2 pixels confidence\r

Example of how to Track a Color with the default mode parameters:

:TC 130 255 0 0 30 30

ACK

M 50 80 38 82 53 128 35 98

M 52 81 38 82 53 128 35 98

TW \r

This command will Track the color found in the central region of the current Window. After the color in the current window is grabbed, the track color function is called with those parameters and on the full screen. This can be useful for locking onto and tracking an object held in front of the camera. Since it actually calls track color, it returns the same type of C or M color packet.

The following internal steps are performed when the “TW” command is called:

1. Shrink the window to 1/4 the size (in each dimension) of the current window. Position the new window at the center of the camera’s field of view. (sw 30 54 50 90)
2. Call the get mean command. (gm)
3. Restore the window to the full image size. (sw 1 1 80 143)
4. Set the min and max value for each color channel to be the mean for that channel +/- 30.

Example of how to use Track Window (Note that Middle Mass mode is not active):

:TW

ACK

S 240 50 40 12 7 8

C 2 40 12 60 10 70

C 2 41 12 61 11 70

OUTPUT DATA PACKET DESCRIPTIONS

All output data packets are in ASCII viewable format except for the F frame and prefix packets.

ACK

This is the standard acknowledge string that indicates that the command was received and fits a known format.

NCK

This is the failure string that is sent when an error occurred. The only time this should be sent when an error has not occurred is during binary data packets.

Type **C** packet:

C x1 y1 x2 y2 pixels confidence\r

This is the return packet from a color tracking command.

x1 - The left most corner's x value

y1 - The left most corner's y value

x2 - The right most corner's x value

y2 -The right most corner's y value

pixels -Number of Pixels in the tracked region, scaled and capped at 255: (pixels+4)/8

confidence -The (# of pixels / area)*256 of the bounded rectangle and capped at 255

Type **F** data packet format:

1 2 r g b r g b ... r g b r g b 2 r g b r g b ... r g b r g b ...

1 - new frame 2 - new col 3 - end of frame

RGB (CrYCb) ranges from 16 - 240

RGB (CrYCb) represents a two pixels color values. Each pixel shares the red and blue.

176 cols of R G B (Cr Y Cb) packets (forms 352 pixels)

144 rows

To display the correct aspect ratio, double each column so that the userr final image is 352x144

It does not begin with an “F” and only sends raw data!

Type **M** packet:

M mx my x1 y1 x2 y2 pixels confidence\r

This is the return packet from a color tracking command with Middle Mass mode on.

mx - The middle of mass x value

my - The middle of mass y value

x1 - The left most corner's x value

y1 - The left most corner's y value

x2 - The right most corner's x value

y2 -The right most corner's y value

pixels –Number of Pixels in the tracked region, scaled and capped at 255: $(pixels+4)/8$

confidence -The $(\# \text{ of pixels} / \text{area}) * 256$ of the bounded rectangle and capped at 255

Type **N** packet:

N spos mx my x1 y1 x2 y2 pixels confidence\r

This is identical to a type **M** packet with an added value for the servo position.

spos – The current position of the servo

Type **S** data packet format:

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

This is a statistic packet that gives information about the camera's view

Rmean - the mean Red or Cr (approximates r-g) value in the current window

Gmean - the mean Green or Y (approximates intensity) value found in the current window

Bmean - the mean Blue or Cb (approximates b-g) found in the current window

Rdeviation - the *deviation of red or Cr found in the current window

Gdeviation- the *deviation of green or Y found in the current window

Bdeviation- the *deviation of blue or Cb found in the current window

*deviation: The mean of the absolute difference between the pixels and the region mean.

Binary bitmap **Line Mode** prefix packet

This packet is in raw byte form only. It starts off with the hex value 0xAA and then streams bytes, with each byte containing a mask for 8 pixels, from the top-left to the bottom-right of the image. (Each binary bit in the byte represents a pixel) The bitmap is then terminated with two 0xAAs. 0xAA is never transmitted as part of the data, so it should be used to signify termination of the binary bitmap. After the binary bitmap is complete, a normal tracking packet should follow.

(raw data: AA XX XX XX XX XX XX AA AA) C 45 72 65 106 18 51

(raw data: AA XX XX XX XX XX XX AA AA) C 46 72 65 106 18 52

Get mean **Line Mode** prefix packet

This packet prefix outputs the mean color of each row being processed. These packets are started with an 0xFE and terminate with an 0xFD. Each byte of data between these values represents the corresponding line's mean color value. Due to the serial transfer time, the vertical resolution is halved. After all rows have been completed, a normal tracking packet should follow.

(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8

(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8

SOFTWARE

MAIN BODY OF THE SOFTWARE

```
#include <avr/io.h>
#include <stdio.h>
#include <math.h>
```

```
#include "PVR.h"
#include "uart.h"
```

```
//Constants
#define ARM_LIMIT 24000
#define SERVOD0_HOME -150
#define SERVOD0_WORK 100
```

```

//Function Declarations
void DriveForward(void);
void DriveBackward(void);
void TurnRight(float time/* in seconds*/,int speed /*inpercentage*/);
void TurnLeft(float time, int speed);
void TurnRandom(void); //Random Turning
FuzzyVar Fuzzify(float crisp); //Fuzzification
float And(float A, float B); // Anding (minimum)
float Or(float A, float B); // Oring (min)
float Not(float A); // Noting(compliment) 1-A;
int Cam2Rob(int cam); // Converts Camera Rotation to robot's equivalence

float Direction(FuzzyVar Right, FuzzyVar Left); //Inference + Defuzzification. Retrurns the drive directions.
void Respond2Bump(void);

void Drive(float direction); //Drives in specified direction.

void Fan(int Swcth); //Turns fan on or off depending on Swcth (1 for on 0 for off)
void ArmUp(int time);
void ArmDown(int time);
int FindBall(); //returns 1 if ball is found and 0 otherwise
void DriveCam();
void Turn(float dir);
void DriveFB(float dir);
void DriveRandom(); //Drive random (ther kernel of state1, implements obstacle avoidance routine)
float absf(float); //bounds an integer by zero
void AlignWithCam();
void delay2Init();
void Centralize(); //Main locking "Guy"
char BallPicked();

// State declarations
void State1(); //Move and Search
void State2(); //Lock on the ball
void State3(); //Pick up the ball
void State4(); //Delivers the ball

TPACKET AlignX();
TPACKET AlignY();

//Speed control variables
int right_speed;
int left_speed;

int right_speed_nominal = 50; //nominal speeds
int left_speed_nominal = 53;
int right_speed, left_speed; //state speeds
float Length = 9.5; //The effective distance between the wheels

//Bump Sensors

```

```

int bump = 0; //state Variable for bump interrupt.
int Bump_Right = 0;
int Bump_Left = 0;
int Bump_Center = 0;
int Bump_Back = 0;

//Fuzzy classification of IR sensor readings
/*float VN = 3125; //13-17cm
float N = 2560; // 17-20 cm
float F = 2133; // 20 - 23 cm
float VF = 1964; //23 - 26cm;
float VVF = 1933; // >26cm
float IRcap = 3615;
float IRcup = 1686; //2220;*/
float VN = 3320; //13-17cm
float N = 1941; // 17-20 cm
float F = 917; // 20 - 23 cm
float VF = 890; //23 - 26cm;
float VVF = 882; // >26cm
float IRcap = 3615;
float IRcup = 300; //2220;

//Arm Control Value

int BallinFocus = 0; // conveys information about the ball being tracked. (0=no ball, 1=ball1, 2 = ball2)
BallColor Ball1;
BallColor Ball2;
char* ball1string = "tc 143 163 57 77 13 23\r";
char* ball2string = "TC 90 120 30 50 5 35\r";
char CameraRX[40];
char no_bytes;

float propGain_angular = 1.2;
float propGain_linear = 0.8; //proportional gain.

float tmax = 2.2;

int CameraPosition = 0; //ranges from -100 to 100
char state = 1; // Control States switches (Defaults to state 1)

void main(void)
{
    //INITIALIZING

    xmegaInit(); //setup XMega
    delayInit(); //setup delay functions
    ServoCInit(); //setup PORTC Servos
    ServoDInit(); //setup PORTD Servos
    ADCAInit(); //setup PORTA analog
    readings

```

```

lcdInit(); //setup LCD on PORTK

uart_init(); // Uart port
delay2Init();

PORTQ_DIR |= 0x01; //set Q0 (LED) as output

PORTB_DIR |= 0xFF;
PORTJ_DIR |= 0xFF;
PORTQ_DIR |= 0xFF;
PORTH_DIR &= 0x00;
PORTF_DIR |= 0xFF; // PORTF used for indicator LEDs.
(0-Startup, 1-State1, 2-State2 3-State 3 4-State 4)

//BUMP INTERRUPTS

PORTCFG_MPCMASK = 0xFF;
PORTH.PINOCTRL = PORT_ISC_FALLING_gc;
PORTH.INTOMASK = 0xFF;
PORTH.INTCTRL = PORT_INT0LVL_HI_gc;
PMIC.CTRL |= PMIC_HILVLEN_bm;

sei();

Fan(0);
PORTJ_OUT |= 0b00001000; // startup;

// Initializing Camera
uart_sendstring("MM 1\r"); //Turns on Mass mode
delay_ms(100); //surpresses Acknowledgement response
uart_sendstring("PM 1\r"); //Engages Pole mode
delay_ms(100);

uart_init();

// Initializing motion parameters
right_speed = left_speed = 0;
right_speed = right_speed_nominal;
left_speed = left_speed_nominal;

while(1)
{
    if(state == 1)
        State1();
    else if(state == 2)
        State2(); //State2: Advances towards the ball
    else if(state == 3)

```

```

        State3();
    else if(state == 4)
        State4();
    }
}

/*****
STATES

State 1: Moves Random and search for ball while achieving obstacle avoidance
Functions Called:
    - DriveRandom()
    - FindBall()

State 2: Achieves first lock on the ball (disregards IR signals)
Functions Called:
    - Respond2Bump():
    - FindBall()
    - Turn()
    - ServoD1()
    - Centralize(): The main lock function: (2D proportional control)

State 3: Tries to pick the ball while keeping lock intact
Functions Called:
    - ServoD0(): Brings the picker to the center
    - ServoC2(): Turns on the vacuum fan
    - FindBall()
    - Centralize()
    - BallPicked(): checks if ball is picked up by taking the arm up and check if camera can
still see the ball

State 4: Delivers the ball home
*****/
void State1()
{
    lcdInit();
    lcdGoto(0,0);
    lcdString("Random Searching ... ");
    PORTJ_OUT |= 0b10000000; //State 1 on
    PORTJ_OUT &= 0b10001111; //States 2,3 and 4 off
    //First SSet BallinFocus
    CameraPosition = FindBall();

    if(BallinFocus==0)
    {
        delaycnt2 = 0; //set count value
        TCD1_CCA = 32000; //set COMA to be 1ms
    delay
        TCD1_CNT = 0; //reset counter
        TCD1_INTCTRLB = 0x01; //enable low priority interruptfor
    delay
}

```

```

        do
        {
            DriveRandom(); //State1 Move random
        }
        while (delaycnt2<3000); //delay
        TCD1_INTCTRLB = 0x00; //disable interrupts
        PORTB_OUT = 0b00101111; //Stoping to allow complete search for the ball.

        CameraPosition = FindBall(); //Try to locate the ball after each move
    }
    else
        state = 2; //Switches to state 2
}
void State2()
{
    TPACKET tpckt;

    lcdInit();
    lcdGoto(0,0);
    lcdString("Ball Found. La La La ...");
    PORTJ_OUT |= 0b01000000; //State 2 on
    PORTJ_OUT &= 0b01001111; //States 1,3 and 4 off

    PORTF_OUT = 0b00000100;
    if(bump) //First respond to bump interrupts
    {
        Respond2Bump();
        bump = 0; // Clear the bump flag;
        CameraPosition = FindBall();
        if(BallinFocus==0)
        {
            state = 1;
            return; //Return back to state 1
        }
        else
        {
            state = 2; //Remain in state 2
            return;
        }
    }
    else if(BallinFocus>0)
    {
        //Locks on the ball

        Centralize();
        ServoD1(0);
        Centralize();
        CameraPosition = 0;
    }
}

```

```

        if(BallinFocus==0)
            state = 1; // Returns to state 1(Ball out of focus)
        else
            state = 3; //Switch to State 3
        //DriveCam();
    }
}
void State3()
{
    lcdInit();
    lcdGoto(0,0);
    lcdString("Lock Achieved. ");
    lcdGoto(1,0);
    lcdString("Trying to Pick. ");

    PORTJ_OUT |= 0b00100000; //State 1 on
    PORTJ_OUT &= 0b00101111; //States 2,3 and 4 off

    ServoD0(SERVOD0_HOME);
    Fan(1);

//    if(PORTJ_OUT!=6)
//    {
        PORTJ_OUT = 0b00000110;           //Start Arm down
        ServoC2(100);

        delaycnt2 = 0;                       //set count value
        TCD1_CCA = 32000;                   //set COMA to be 1ms
    delay

        TCD1_CNT = 0;                       //reset counter
        TCD1_INTCTRLB = 0x01;               //enable low priority interruptfor
    delay

//    }

//    while(delaycnt2<(ARM_LIMIT)) // Keeps lock for 1 second after arm gets down to allow enough time for
//    picking ball
//    {
//        Centralize();
//    }

// Return Arm up
    delay_ms(ARM_LIMIT+5000);
    delaycnt2 = 0;
    PORTJ_OUT = 0b00000101;

//    delay_ms(ARM_LIMIT); //delays for 3/5 of total time taken to get up finally to allow enough time for ball
//    to get out fo camera view

//    //Check for ball again
//    CameraPosition = FindBall();

```

```

//      if(BallinFocus == 0)
//          state = 4; //Ball not in focus, assume picked and go to state 4
//      else
//          {
//              while(delaycnt2<ARM_LIMIT);
//              state = 1; //Ball in focus, go back to state 1 to start all over again
//          }
//      lcdGoto(0,0);
//      lcdInt(state);
state = 4;

}
void State4()
{
    lcdInit();
    lcdGoto(0,0);
    lcdString("Yeah !");
    lcdGoto(1,0);
    lcdString("Delivering Ball");

    PORTJ_OUT |= 0b00010000; //State 4 on
    PORTJ_OUT &= 0b00011111; //States 1,2 and 3 off

    // checks for complete journey of arm up
    while(delaycnt2<(ARM_LIMIT+2000));

    PORTJ_OUT = 0b00000111; //Stops Arm motion
    TCD1_INTCTRLB = 0x00; //disable interrupts

    //Move servoD0 to work
    ServoD0(SERVOD0_WORK);

    //DropBall
    Fan(0);
    delay_ms(3000);

    //Move servoD0 back to home
    ServoD0(SERVOD0_HOME);

    state = 1; //switch back to state 1;
}

/*****
END OF STATES
*****/
char BallPicked()
{
    //Move arm up
    ArmUp(ARM_LIMIT);
    char CameraRX2[4];

```

```

//Polls for the ball
do{
    uart_sendstring(ball1string);
    no_bytes = uart_getstring(CameraRX2);
    no_bytes = uart_getstring(CameraRX);
}
while(CameraRX2[1]!='N');

TPACKET tpckt;
tpckt = tpacketDecode(CameraRX);

if(tpckt.confidence<8)
{
    BallinFocus = 0;
    return 1;
}
else
{
    BallinFocus = 1;
    return 0;
}
}

```

```

void Centralize()
{
    float error_angular;
    TPACKET tpckt;
    float tol = 4;

    //Rotational Compensation
    do{
        tpckt = AlignX();
        if(tpckt.confidence<8)
        {
            BallinFocus = 0;
            break;
        }
        tpckt = AlignY();
        if(tpckt.confidence<8)
        {
            BallinFocus = 0;
            break;
        }
    }
    /* error_angular= tpckt.mx-50;
    lcdInit();
    lcdGoto(0,0);
    lcdInt(error_angular);*/

    //Translational Compensation
}

```

```

        while(absf(error_angular)>tol);

        return;
    }
    TPACKET AlignX()
    {
        char CameraRX2[4];
        float error_angular;
        TPACKET tpckt;
        float tol = 2;
        do{
            do{
                uart_sendstring(ball1string);
                no_bytes = uart_getstring(CameraRX2);
                no_bytes = uart_getstring(CameraRX);
            }
            while(CameraRX2[1]!='N');

            tpckt = tpacketDecode1(CameraRX,no_bytes);

            if(tpckt.confidence<8)
            {
                BallinFocus = 0;
                return tpckt;
            }

            error_angular= tpckt.mx-50;
            Turn(propGain_angular*error_angular);

        }
        while(absf(error_angular)>tol);

        return tpckt;
    }

    TPACKET AlignY()
    {lcdInit();
        char CameraRX2[4];
        float error_linear;
        TPACKET tpckt;
        float tol = 6;
        do{

            do{
                uart_sendstring(ball1string);
                no_bytes = uart_getstring(CameraRX2);
                no_bytes = uart_getstring(CameraRX);
            }

```

```

        while(CameraRX2[1]!='N');

        tpckt = tpacketDecode1(CameraRX,no_bytes);

        if(tpckt.confidence<8)
        {
            BallinFocus = 0;
            return tpckt;
        }
        lcdGoto(0,0);
        lcdString("Here 111111111111");
        error_linear= 100-tpckt.my;

        DriveFB(propGain_linear*error_linear);

    }
    while(absf(error_linear)>tol);

    return tpckt;
}

void SetBallRGB(BallColor* B1, BallColor* B2)
{
    //Displaying Dialog
    lcdInit();
    lcdGoto(0,0);
    lcdString("Calib Camera");
    lcdGoto(1,0);
    lcdString("Ball 1");
    delay_ms(3000);

    for(int i=0;i<10;i++)
    {
        uart_sendstring("GM\r");
        no_bytes = uart_getstring(CameraRX);
        no_bytes = uart_getstring(CameraRX);
        lcdGoto(1,0);

    }

    SPACKET spckt;
    spckt = spacketDecode(CameraRX);
    B1->R = spckt.r;
    B1->B = spckt.b;
    B1->G = spckt.g;

    lcdInit();
    lcdGoto(0,0);
    lcdString("Calib Camera");
    lcdGoto(1,0);

```

```

    lcdString("Ball 2");
    delay_ms(3000);
    for(int i=0;i<10;i++)
    {
        uart_sendstring("GM\r");
        no_bytes = uart_getstring(CameraRX);
        no_bytes = uart_getstring(CameraRX);
        lcdGoto(1,0);
    }

    spckt = spacketDecode(CameraRX);
    B2->R = spckt.r;
    B2->B = spckt.b;
    B2->G = spckt.g;

    lcdInit();
    lcdGoto(0,0);
    lcdString("Calib Done");

}
void delay2Init(void)
{
    TCD1_CTRLA = 0x01; //set clock/1
    TCD1_CTRLB = 0x31; //enable COMA and COMB, set to
FRQ
    TCD1_INTCTRLB = 0x00; //turn off interrupts for COMA and COMB
    SREG |= CPU_I_bm; //enable all interrupts
    PMIC_CTRL |= 0x01; //enable all low priority interrupts
}
SIGNAL(TCD1_CCB_vect)
{
    delaycnt2++;
    if(delaycnt2>=ARM_LIMIT)
    {
        PORTJ_OUT = 0b00000111; //Stop Arm down;
    }
}

SIGNAL(TCD1_CCA_vect)
{
    delaycnt2++;
    if(delaycnt2>=ARM_LIMIT)
    {
        PORTJ_OUT = 0b00000111; //Stop Arm down;
    }
}

int FindBall()
{
    BallinFocus = 0;

```

```

int pos = 0,i = 1,count;

char CameraRX2[4];
TPACKET tpckt;

do
{
    //TrackBall
    count = 0;

    do
    {
        PORTF_OUT = 0b00000010;
        uart_sendstring(ball1string);

        no_bytes = uart_getstring(CameraRX2);
        no_bytes = uart_getstring(CameraRX);
        count++;
    }
    while(CameraRX2[1]!='N'&&(count<3));

    tpckt = tpacketDecode1(CameraRX,no_bytes);
    if(tpckt.confidence>=40)
    {
        BallinFocus = 1;
        break;
    }
    else
    {
        pos = i*10-100;
        ServoD1(pos);
        i++;
    }
}
while(i<20&&tpckt.confidence<30);

return pos;
}
/*
int pos = 0;
char CameraRX2[40];
if(BallinFocus==0) //No ball in focus, has to search all directions for balls 1 and 2
{
    for(int i = 0;i<=20;i++)
    {
        pos = i*10-100;
        ServoD1(pos);

        //Tracking Ball1
        char* s1;
        s1 = strcat("TC ",itoa(Pos(Ball1.R-30)));
        s1 = strcat(s1, " ");
    }
}

```

```

s1 = strcat(s1,itoa(Ball1.R+30));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Pos(Ball1.G-30)));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Ball1.G+30));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Pos(Ball1.B-30)));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Ball1.B+30));

uart_sendstring(strcat(s1,"\r"));
//for(int k=0;k<5;k++){
//do{
uart_sendstring(ball1string);
no_bytes = uart_getstring(CameraRX2);
no_bytes = uart_getstring(CameraRX);}while(CameraRX2[1]!='N');

LcdInit();
LcdGoto(0,0);
LcdChar(CameraRX2[1]);
LcdGoto(1,0);
LcdInt(pos);

TPACKET tpckt;
tpckt = tpacketDecode(CameraRX);

if(tpckt.confidence>=50)
{
    BallinFocus = 1;
    LcdGoto(1,0);
    return pos;
}

//Tracking Ball2
s1 = "";
s1 = strcat("TC ",itoa(Pos(Ball2.R-30)));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Ball2.R+30));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Pos(Ball2.G-30)));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Ball2.G+30));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Pos(Ball2.B-30)));
s1 = strcat(s1, " ");
s1 = strcat(s1,itoa(Ball2.B+30));

uart_sendstring(strcat(s1,"\r"));
do{
uart_sendstring(ball2string);
no_bytes = uart_getstring(CameraRX2);
no_bytes = uart_getstring(CameraRX);}while(CameraRX2[1]!='N');

```

```

        tpckt = tpacketDecode(CameraRX);

        if(tpckt.confidence>=50)
        {
            BallinFocus = 2;
            return pos;
        }
    }

    return pos;
}
*/

int Cam2Rob(int cam)
{
    float res = cam*1.0952 - 0.57143;
    return res;
}

float absf(float val)
{
    if(val<0)
        val*=-1;

    return val;
}

void DriveCam()
{
    delaycnt2 = 0; //set count value
    TCD1_CCA = 32000; //set COMA to be 1ms delay
    TCD1_CNT = 0; //reset counter
    TCD1_INTCTRLB = 0x01; //enable low priority interruptfor delay

    lcdInit();
    lcdGoto(0,0);
    lcdString("Inside Drive Cam");

    char CameraRX2[4];
    int IR_value;

    do{
        do{
            uart_sendstring(ball1string);
            no_bytes = uart_getstring(CameraRX2);
            no_bytes = uart_getstring(CameraRX);
        }
        while(CameraRX2[1]!='N');
    }
}

```

```

TPACKET tpckt;
tpckt = tpacketDecode(CameraRX);

if(tpckt.confidence<=10)
{
    BallinFocus = 0;
}

else if(BallinFocus!=2)
{

    float left_speed = 20;//(tpckt.my-40)*propGain1; //constant left speed

    float right_speed =20;// left_speed + (tpckt.mx-50)*propGain2; //50 percent speed

    PORTB_OUT = 0b00101010;
    lcdGoto(1,0);
    lcdInt(left_speed);
    lcdGoto(1,5);
    lcdInt((tpckt.mx-50)*propGain_angular);
    ServoC4(right_speed);
    ServoC5(left_speed);

}

IR_value = ADCA1();
lcdInit();
lcdGoto(0,0);
lcdInt(IR_value);

if(IR_value>=1500&&IR_value<=1700)
{
    BallinFocus = 2; //Set to Picking up mode;

    lcdGoto(0,0);
    lcdString("Ready to Pick");
    PORTB_OUT = 0b00101111; //Stop the movement;

    break;
}
}while(delaycnt2<5000);
PORTB_OUT = 0b00101111; //Stop the movement;

TCD1_INTCTRLB = 0x00; //disable interrupts

}

void DriveRandom()
{
    //PORTF_OUT = 0b00000010;

```

```

        //IR Fuzzy variables
FuzzyVar IR_Right,
        IR_Left,
        IR_Center;

/*    PORTA_DIR &= 0b11111110;
    IRcalib = ADCA0();
    lcdGoto(0,0);
    lcdInt(IRcalib);*/

    PORTA_DIR &= 0b11111110;
    //ADCAInit();
    IR_Right = Fuzzify(ADCA0());
    PORTA_DIR &= 0b11111101;
    //ADCAInit();
    IR_Center = Fuzzify(ADCA1());
    PORTA_DIR &= 0b11111011;
    //ADCAInit();
    IR_Left = Fuzzify(ADCA2());

/*    lcdInit();

    lcdGoto(0,0);
    lcdInt(100*IR_Center.VN);
    lcdGoto(0,6);
    lcdInt(100*IR_Center.N);
    lcdGoto(0,12);
    lcdInt(100*IR_Center.F);

    lcdGoto(1,0);
    lcdInt(100*IR_Center.VF);
    lcdGoto(1,6);
    lcdInt(100*IR_Center.VVF);
    lcdGoto(1,12);
    lcdInt(ADCA1());*/

    if(bump)
    {
        //lcdGoto(0,0);
        //lcdString("1");
        Respond2Bump();
        bump = 0; // Clear the bump flag;
    }
    else if(IR_Center.VN>=80)
    {
        //lcdGoto(0,0);
        //lcdString("2");
        DriveBackward();
        delay_ms(1000);
        TurnRandom();
    }
}

```

```

        else{
            //    lcdGoto(0,0);
            //    lcdString("3");
            Drive(Direction(IR_Right,IR_Left));
        }
        //delay_ms(100);
        //PORTF_OUT = 0b00000000;
    }

void Fan(int Swtch)
{
    if(Swtch)
    {
        PORTQ_OUT &= 0xFE;
    }
    else
        PORTQ_OUT |= 0x01;
}

float Direction(FuzzyVar R, FuzzyVar L)
{
    //Inference
    float HR,HL,Le,Ri,S;

    /* HR -> Hard Right (100)
       HL -> Hard Left (-100)
       Ri -> Right (50)
       Le -> Left (-50);
       S -> Straight (0)
       */

    HL = Or(And(R.VN,Or(L.VN,Or(L.N,Or(L.F,Or(L.VF,L.VVF))))),And(L.F,R.N));
    Le = Or(And(L.N,R.N),Or(And(L.F,R.F),And(R.N,Or(L.VF,L.VVF))));
    S = Or(And(Or(L.VF,L.VVF),Or(R.F,Or(R.VF,R.VVF))),And(L.F,Or(R.VF,R.VVF)));
    HR = Or(And(L.VN,Or(R.N,Or(R.F,Or(R.VF,R.VVF))))),And(L.N,R.F));
    Ri = And(L.N,Or(R.VF,R.VVF));

    //Defuzzification (weighted Average)
    /*lcdGoto(0,0);
    lcdInt(HR);
    lcdGoto(0,5);
    lcdInt(Ri);
    lcdGoto(0,10);
    lcdInt(S);
    lcdGoto(1,0);
    lcdInt(Le);
    lcdGoto(1,5);
    lcdInt(HL);*/

    return(HR-HL+0.50*Ri-.50*Le); //There is no need to calculate for S(Straight). Since it's weight is 0
}

```

```

float And(float A, float B)
{
    if(A<=B)
        return A;
    else
        return B;
}
float Or(float A, float B)
{
    if(A>=B)
        return A;
    else
        return B;
}
float Not(float A)
{
    return (1-A);
}
FuzzyVar Fuzzify(float crisp)
{
    FuzzyVar FV;
    FV.VN=0;
    FV.N = 0;
    FV.F = 0;
    FV.VF = 0;
    FV.VVF = 0;

    if(crisp<=IRcap && crisp > N) //Very near
    {
        if(crisp>VN)
            FV.VN = 100;
        else
            FV.VN = 100*(crisp-N)/(VN-N); //100*(1 -(VN-crisp)/(VN-N));
    }

    if(crisp<=VN && crisp >F) //Near
    {
        if(crisp>N)
            FV.N = 100*(VN-crisp)/(VN-N);
        else
            FV.N = 100*(crisp-F)/(N-F); //100*(1 - (N-crisp)/(N-F));
    }

    if(crisp<=N && crisp >VF) //Far
    {
        if(crisp>F)
            FV.F = 100*(N-crisp)/(N-F);
        else
            FV.F = 100*(crisp-VF)/(F-VF); //100*(1 - (F-crisp)/(F-VF));
    }

    if(crisp<=F && crisp >VVF) // Very Far
    {

```

```

        if(crisp>VF)
            FV.VF = 100*(F-crisp)/(F-VF);
        else
            FV.VF = 100*(crisp-VVF)/(VF-VVF);//100*(1 - (VF-crisp)/(VF-VVF));
    }

    if(crisp<VF && crisp>IRcup)
    {
        if(crisp<VVF)
            FV.VVF = 100;
        else
            FV.VVF = 100*(VF-crisp)/(VF-VVF);
    }

    return FV;
}

```

```

/*SIGNAL(USARTE1_RXC_vect)
{

}*/

```

```

SIGNAL(PORTH_INT0_vect)
{
    bump = 1;
    if(!(PORTH_IN&0b00000001))
    {
        Bump_Back = 1;
    }
    else if(!(PORTH_IN&0b00000010))
    {
        Bump_Left = 1;
    }
    else if(!(PORTH_IN&0b00000100))
    {
        Bump_Center = 1;
    }
    else if(!(PORTH_IN&0b0001000))
    {
        Bump_Right = 1;
    }

    cli();
}

```

```

void Respond2Bump(void)
{

    if(Bump_Center)
    {

```

```

        DriveBackward();
        delay_ms(1000);
        TurnRandom();
        Bump_Center = 0; //Resetting Flag
    //    lcdGoto(1,0);
    //    lcdString("BumpCenter");
    }
    else if(Bump_Right)
    {
        delay_ms(500);
        DriveBackward();
        TurnLeft(2,40);
        Bump_Right = 0; //Resetting Flag;
    //    lcdGoto(1,0);
    //    lcdString("BumpRight");
    }
    else if(Bump_Left)
    {
        DriveBackward();
        delay_ms(500);
        TurnRight(2,40);
        Bump_Left = 0; //Resetting Flag;
    //    lcdGoto(1,0);
    ///    lcdString("BumpLeft");
    }

    if(Bump_Back)
    {
        DriveForward();
        delay_ms(200);
        Bump_Back = 0; //resetting flag
    //    lcdGoto(1,0);
    //    lcdString("BumpBack");
    }
    sei();
}

void Turn(float dir)
{
    //    float delta = 45*dir*Length/427;
    //    left_speed -= delta;
    //    right_speed += delta;
    if(dir<0)
    {
        PORTB_OUT |=0b00000100;
        PORTB_OUT &=0b11110111;
        dir*=-1;
    }
    else//if(left_speed<0)
    {
        PORTB_OUT |=0b00000001;
        PORTB_OUT &=0b11111101;
    }
    //if(dir<0)

```

```

        //    dir*=-1;

        ServoC4(50);
        ServoC5(50);

        delay_ms(10*dir);
        PORTB_OUT = 0b00101111;
    }

void Drive(float dir)
{
    //DIR: [-100 100] DETERMINES THE DRIVE DIRECTION
    if(dir==0) //Drive strigth
        DriveForward();
    else if(dir>100 | dir<-100)
        DriveBackward();
/*    else
        Turn(dir);*/

    else if(dir<0)
    {
        TurnLeft(-1*dir*tmax/100,40);
    }
    else if(dir>0)
    {
        TurnRight(dir*tmax/100,40);
    }
}

void DriveFB(float del)
{
    if(del<0)
    {
        del*=-1;
        PORTB_OUT = 0b00101111;
        PORTB_OUT = 0b00100101;
    }
    else
    {
        PORTB_OUT = 0b00101111;
        PORTB_OUT = 0b00101010;
    }

    ServoC4(100);
    ServoC5(100);

    delay_ms(10*del);
    PORTB_OUT = 0b00101111;
}

void DriveForward()

```

```

{
    PORTB_OUT = 0b00101010;
    ServoC4(right_speed);
    ServoC5(left_speed);
}

void DriveBackward()
{
    //Stop First
    PORTB_OUT = 0b00101111;
    PORTB_OUT = 0b00100101;
    ServoC4(right_speed);
    ServoC5(left_speed);
}

void TurnRight(float time,int speed)
{
    //Stop First
    //PORTB_OUT = 0b00101111;

    ServoC5(speed);
    ServoC4(speed);
    PORTB_OUT = 0b00100110;
    delay_ms(time*500);
    PORTB_OUT = 0b00101111;
}

void TurnLeft(float time,int speed /* angular speed */)
{
    //Stop First
    //PORTB_OUT = 0b00101111;
    ServoC5(speed);
    ServoC4(speed);
    PORTB_OUT = 0b00101001;
    delay_ms(time*500);
    PORTB_OUT = 0b00101111;
}

void TurnRandom()
{
    if(TCF1_CNT & 0x0010)
    {
        TurnRight(1,40);
    }
    else
    {
        TurnLeft(1,40);
    }
}

```

```

        TCF1_CNT = 0;
    }

void ArmUp(int time)
{
    PORTJ_OUT = 0b00000101;
    ServoC2(100);
    delay_ms(time); //Motor is faster moving down.
    PORTJ_OUT = 0b00000111;
}

void ArmDown(int time)
{
    PORTJ_OUT = 0b00000110;
    ServoC2(100);
    delay_ms(time);
    PORTJ_OUT = 0b00000111;
}

```

PVR.C (FROM MIKE AND THOMAS, TEACHING ASSISTANTS)

```

#include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR.h"

/*****
 * Xmega *
 *****/

void xmegaInit(void)
{
    CCP = 0xD8;
    CLK_PSCTRL = 0x00;
    PORTQ_DIR = 0x01;
    //setup oscillator
    OSC_CTRL = 0x02;
    while ((OSC_STATUS & 0x02) == 0); //enable 32MHz internal clock
    CCP = 0xD8; //wait for oscillator to be ready
    CLK_CTRL = 0x01; //write signature to CCP
    //select internal 32MHz RC oscillator
}

/*****
 * Delay *
 *****/

void delayInit(void)
{
    TCF1_CTRLA = 0x01; //set clock/1
    TCF1_CTRLB = 0x31; //enable COMA and COMB, set to
FRQ
    TCF1_INTCTRLB = 0x00; //turn off interrupts for COMA and COMB
    SREG |= CPU_I_bm; //enable all interrupts
}

```

```

        PMIC_CTRL |= 0x01;                //enable all low priority interrupts
    }

void delay_ms(int cnt)
{
    delaycnt = 0;                        //set count value
    TCF1_CCA = 32000;                    //set COMA to be 1ms delay
    TCF1_CNT = 0;                        //reset counter
    TCF1_INTCTRLB = 0x01;                //enable low priority interrupt for delay
    while (cnt != delaycnt);             //delay
    TCF1_INTCTRLB = 0x00;                //disable interrupts
}

void delay_us(int cnt)
{
    delaycnt = 0;                        //set counter
    TCF1_CCA = 32;                       //set COMA to be 1us delay
    TCF1_CNT = 0;                        //reset counter
    TCF1_INTCTRLB = 0x01;                //enable low priority interrupt for delay
    while (cnt != delaycnt);             //delay
    TCF1_INTCTRLB = 0x00;                //disable interrupts
}

SIGNAL(TCF1_CCB_vect)
{
    delaycnt++;
    TimerBValue++;
}

SIGNAL(TCF1_CCA_vect)
{
    delaycnt++;
}

/*****
 * LCD *
*****/

#define LCD          PORTK_OUT
#define LCDDDR      PORTK_DIR

void lcdDataWork(unsigned char c)
{
    c &= 0xF0;                            //keep data bits, clear the
rest
    c |= 0x08;                             //set E high
    LCD = c;                               //write to LCD
    delay_ms(2);                           //delay
    c ^= 0x08;                             //set E low
    LCD = c;                               //write to LCD
    delay_ms(2);                           //delay
    c |= 0x08;                             //set E high
    LCD = c;                               //write to LCD
}

```

```

        delay_ms(2);                //delay
    }

void lcdData(unsigned char c)
{
    unsigned char cHi = c & 0xF0;    //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;    //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                //shift cLo left 4 bits
    lcdDataWork(cHi);
    lcdDataWork(cLo);
}

void lcdCharWork(unsigned char c)
{
    c &= 0xF0;                        //keep data bits, clear the
rest                                  //set E and RS high
    c |= 0x0A;                          //write to LCD
    LCD = c;                              //delay
    delay_ms(2);                          //set E low
    c ^= 0x08;                            //write to LCD
    LCD = c;                              //delay
    delay_ms(2);                          //set E high
    c |= 0x08;                            //write to LCD
    LCD = c;                              //delay
    delay_ms(2);                          //delay
}

void lcdChar(unsigned char c)
{
    unsigned char cHi = c & 0xF0;    //give cHi the high 4 bits of c
    unsigned char cLo = c & 0x0F;    //give cLo the low 4 bits of c
    cLo = cLo * 0x10;                //shift cLo left 4 bits
    lcdCharWork(cHi);
    lcdCharWork(cLo);
}

void lcdString(unsigned char ca[])
{
    int i = 0;
    while (ca[i] != '\0')
    {
        lcdChar(ca[i++]);
    }
}

void lcdInt(int value)
{
    int temp_val;
    int x = 10000;
    int leftZeros=5;

    if (value<0)

```

```

        {
            lcdChar('-');
            value *= -1;
        }

while (value / x == 0)
{
    x/=10;
    leftZeros--;
}

while ((value > 0) || (leftZeros>0))
{
    temp_val = value / x;
    value -= temp_val * x;
    lcdChar(temp_val+ 0x30);
    x /= 10;
    leftZeros--;
}

while (leftZeros>0)
{
    lcdChar(0+ 0x30);
    leftZeros--;
}

return;
}

void lcdGoto(int row, int col)
{
    unsigned char pos;
    if ((col >= 0 && col <= 19) && (row >= 0 && row <= 3))
    {
        pos = col;
        if (row == 1)
            pos += 0x40;
        else if (row == 2)
            pos += 0x14;
        else if (row == 3)
            pos += 0x54;
        lcdData(0x80 + pos);
    }
}

void lcdInit(void)
{
    delayInit(); //set up the delay functions
    LCDDDR = 0xFF; //set LCD port to outputs.
    delay_ms(20); //wait to ensure LCD powered up
    lcdDataWork(0x30); //put in 4 bit mode, part 1
    delay_ms(10); //wait for lcd to finish
}

```

```

        lcdDataWork(0x30);
        delay_ms(2);
        lcdData(0x32);
        lcdData(0x2C);
        lcdData(0x0C);
        lcdData(0x01);
    }

    /*****
    * Servo *
    *****/

    void ServoCInit(void)
    {
        TCC0_CTRLA = 0x05;
        TCC0_CTRLB = 0xF3;

        and 0 from CCx to Top
        TCC0_PER = 10000;
        TCC1_CTRLA = 0x05;
        TCC1_CTRLB = 0x33;

        and 0 from CCx to Top
        TCC1_PER = 10000;
        PORTC_DIR = 0x3F;
        TCC0_CCA = 0;
        TCC0_CCB = 0;
        TCC0_CCC = 0;
        TCC0_CCD = 0;
        TCC1_CCA = 0;
        TCC1_CCB = 0;
    }

    void ServoDInit(void)
    {
        TCD0_CTRLA = 0x05;
        TCD0_CTRLB = 0xF3;

        and 0 from CCx to Top
        TCD0_PER = 10000;
        TCD1_CTRLA = 0x05;
        TCD1_CTRLB = 0x33;

        and 0 from CCx to Top
        TCD1_PER = 10000;
        PORTD_DIR = 0x3F;
        TCD0_CCA = 0;
        TCD0_CCB = 0;
        TCD0_CCC = 0;
        TCD0_CCD = 0;
        TCD1_CCA = 0;
        TCD1_CCB = 0;
    }

    //put in 4 bit mode, part 2
    //wait for lcd to finish
    //put in 4 bit mode, part 3
    //enable 2 line mode
    //turn everything on
    //clear LCD
    //set TCC0_CLK to CLK/64
    //Enable OC A, B, C, and D. Set to Single Slope PWM
    //OCnX = 1 from Bottom to CCx
    //20ms / (1/(32MHz/64)) = 10000. PER = Top
    //set TCC1_CLK to CLK/64
    //Enable OC A and B. Set to Single Slope PWM
    //OCnX = 1 from Bottom to CCx
    //20ms / (1/(32MHz/64)) = 10000. PER = Top
    //set PORTC5:0 to output
    //PWMC0 off
    //PWMC1 off
    //PWMC2 off
    //PWMC3 off
    //PWMC4 off
    //PWMC5 off
    //set TCC0_CLK to CLK/64
    //Enable OC A, B, C, and D. Set to Single Slope PWM
    //OCnX = 1 from Bottom to CCx
    //20ms / (1/(32MHz/64)) = 10000. PER = Top
    //set TCC1_CLK to CLK/64
    //Enable OC A and B. Set to Single Slope PWM
    //OCnX = 1 from Bottom to CCx
    //20ms / (1/(32MHz/64)) = 10000. PER = Top
    //set PORTC5:0 to output
    //PWMC0 off
    //PWMC1 off
    //PWMC2 off
    //PWMC3 off
    //PWMC4 off
    //PWMC5 off

```

```

void ServoC0(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                // -100 => 1ms
    else if (value < -100)          // 0  => 1.5ms
        value = -100;               // 100 => 2ms
    value *= 5;                      //multiply value by 2.5
    value /= 2;                      // new range +/- 250
    TCCO_CCA = (750 + value);        //Generate PWM.
}

```

```

void ServoC1(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                // -100 => 1ms
    else if (value < -100)          // 0  => 1.5ms
        value = -100;               // 100 => 2ms
    value *= 5;                      //multiply value by 2.5
    value /= 2;                      // new range +/- 250
    TCCO_CCB = (750 + value);        //Generate PWM.
}

```

/******

ServoC3, ServoC4 and ServoC5 are used to drive the wheel DC motors */

```

void ServoC2(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                // -100 => 1ms
    else if (value < 0)             // 0  => 1.5ms
        value = 0;                  // 100 => 2ms

    TCCO_CCC = 100*value;           //Generate PWM.
}

```

```

void ServoC3(int value)
{
    if (value > 100)                //cap at +/- 100
        value = 100;                // -100 => 1ms
    else if (value < 0)             // 0  => 1.5ms
        value = 0;                  // 100 => 2ms
    // new range +/- 250

    TCCO_CCD = 100*value;           //Generate PWM.
}

```

```

void ServoC4(int value)
{
    if (value > 100)                //cap at [0 100] percentage
        value = 100;                // 100 => 20ms
    else if (value < 0)             // 0  => 0ms
        value = 0;
}

```

```

        TCC1_CCA = 100*value;          //Generate PWM.
    }

void ServoC5(int value)
{
    if (value > 100)                  //cap at [0 100] percentage
        value = 100;                  // 100 => 20ms
    else if (value < 0)                // 0  => 0ms
        value = 0;

    TCC1_CCB = 100*value;            //Generate PWM.
}
/*****

void ServoD0(int value)
{
    if (value > 200)                  //cap at +/- 100
        value = 200;                  // -100 => 1ms
    else if (value < -200)            // 0  => 1.5ms
        value = -150;                 // 100 => 2ms
    value *= 5;                       //multiply value by 2.5
    value /= 2;                       // new range +/- 250
    TCDO_CCA = (750 + value);         //Generate PWM.
}

void ServoD1(int value)
{
    if (value > 100)                  //cap at +/- 100
        value = 100;                  // -100 => 1ms
    else if (value < -100)            // 0  => 1.5ms
        value = -100;                 // 100 => 2ms
    value *= 5;                       //multiply value by 2.5
    value /= 2;                       // new range +/- 250
    TCDO_CCB = (750 + value);         //Generate PWM.
}

void ServoD2(int value)
{
    if (value > 100)                  //cap at +/- 100
        value = 100;                  // -100 => 1ms
    else if (value < -100)            // 0  => 1.5ms
        value = -100;                 // 100 => 2ms
    value *= 5;                       //multiply value by 2.5
    value /= 2;                       // new range +/- 250
    TCDO_CCC = (750 + value);         //Generate PWM.
}

void ServoD3(int value)
{
    if (value > 100)                  //cap at +/- 100
        value = 100;                  // -100 => 1ms
    else if (value < -100)            // 0  => 1.5ms

```

```

        value = -100;                // 100 => 2ms
    value *= 5;                      //multiply value by 2.5
    value /= 2;                      // new range +/- 250
    TCD0_CCD = (750 + value);        //Generate PWM.
}

void ServoD4(int value)
{
    if (value > 100)                 //cap at +/- 100
        value = 100;                // -100 => 1ms
    else if (value < -100)           // 0  => 1.5ms
        value = -100;              // 100 => 2ms
    value *= 5;                      //multiply value by 2.5
    value /= 2;                      // new range +/- 250
    TCD1_CCA = (750 + value);        //Generate PWM.
}

void ServoD5(int value)
{
    if (value > 100)                 //cap at +/- 100
        value = 100;                // -100 => 1ms
    else if (value < -100)           // 0  => 1.5ms
        value = -100;              // 100 => 2ms
    value *= 5;                      //multiply value by 2.5
    value /= 2;                      // new range +/- 250
    TCD1_CCB = (750 + value);        //Generate PWM.
}

/*****
* ADCA *
*****/

void ADCAInit(void)
{
    ADCA_CTRLB = 0x00;                //12bit, right adjusted
    ADCA_REFCTRL = 0x10;              //set to Vref = Vcc/1.6 = 2.0V (approx)
    ADCA_CHO_CTRL = 0x01;            //set to single-ended
    ADCA_CHO_INTCTRL = 0x00;         //set flag at conversion complete. Disable interrupt
    ADCA_CHO_MUXCTRL = 0x08;         //set to Channel 1
    ADCA_CTRLA |= 0x01;              //Enable ADCA
}

int ADCA0(void)
{
    ADCA_CHO_MUXCTRL = 0x00;          //Set to Pin 0
    delay_ms(1);
    ADCA_CTRLA |= 0x04;              //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;         //grab result
    return value;                    //return result
}

```

```

int ADCA1(void)
{
    ADCA_CHO_MUXCTRL = 0x08;           //Set to Pin 1
    delay_ms(1);
    ADCA_CTRLA |= 0x04;                //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;          //grab result
    return value;                      //return result
}

int ADCA2(void)
{
    ADCA_CHO_MUXCTRL = 0x10;           //Set to Pin 2
    delay_ms(1);
    ADCA_CTRLA |= 0x04;                //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;          //grab result
    return value;                      //return result
}

int ADCA3(void)
{
    ADCA_CHO_MUXCTRL = 0x18;           //Set to Pin 3
    delay_ms(1);
    ADCA_CTRLA |= 0x04;                //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;          //grab result
    return value;                      //return result
}

int ADCA4(void)
{
    ADCA_CHO_MUXCTRL = 0x20;           //Set to Pin 4
    delay_ms(1);
    ADCA_CTRLA |= 0x04;                //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;          //grab result
    return value;                      //return result
}

int ADCA5(void)
{
    ADCA_CHO_MUXCTRL = 0x28;           //Set to Pin 5
    delay_ms(1);
    ADCA_CTRLA |= 0x04;                //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;          //grab result
    return value;                      //return result
}

```

```

}

int ADCA6(void)
{
    ADCA_CHO_MUXCTRL = 0x30;           //Set to Pin 6
    delay_ms(1);
    ADCA_CTRLA |= 0x04;                //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;          //grab result
    return value;                       //return result
}

int ADCA7(void)
{
    ADCA_CHO_MUXCTRL = 0x38;           //Set to Pin 7
    delay_ms(1);
    ADCA_CTRLA |= 0x04;                //Start Conversion on ADCA Channel 0
    while ((ADCA_CHO_INTFLAGS & 0x01) != 0x01); //wait for conversion to complete
    delay_ms(1);
    int value = ADCA_CHO_RES;          //grab result
    return value;                       //return result
}

char* substr(char* str,int fro,int to)
{
    char* result = "";
    int j = 0;
    for(int i = fro;i<=to;i++)
    {
        result[j] = str[i];
        j++;
    }

    return result;
}

```

UART.C (ORIGINALLY FROM HAO HE BUT MODIFIED BY ME TO WORK WITH PINGBOT)

```

#include <avr/io.h>
#include <string.h>
#include <stdlib.h>
#include "PVR.h"
#include "uart.h"

void uart_init(void)
{
    PORTE_OUT  |= 0b10000000;
    PORTE_DIR  |= 0b10000000; //PIN3 as output
    PORTE_DIR &= 0b10111111; //PIN2 as input
}

```

```

PORTE_OUT &= 0b10111111;

USARTE1_BAUDCTRLA = 0x10;           // set to BSEL = 0x10, BSCALE = 0x00
USARTE1_BAUDCTRLB = 0x00;           // fbaud = 32MHz/(1*(16+1)*16)= 117,647 bps
// USARTE1_BAUDCTRLA = 0x0C; //BSEL = 12, BSCALE = 4, fBaud = 9600
// USARTE1_BAUDCTRLB = 0x40;

USARTE1_CTRLA = 0b00000011;         // set 8N1 asynchronous serial tx/rx
USARTE1_CTRLB = 0b00011000;
}

char uart_getchar(void)
{
    char rx_char;                    // initialize received character buffer

    while (!(USARTE1_STATUS & 0b10000000)); // wait for RXCIF to be set

    rx_char = USARTE1_DATA;           // data register in variable rx_char

    return rx_char;
}

void uart_sendchar(char tx_char)
{
    while (!(USARTE1_STATUS & (1<<USART_DREIF_bp))); // check if data register is empty

    USARTE1_DATA = tx_char;           // store data in
tx_char

    while (!(USARTE1_STATUS & (1<<USART_TXCIF_bp))); // wait for RXCIF to be sent
}

void uart_sendstring(char tx_string[])
{
    int i = 0;
    while (tx_string[i] != '\r')      //while not line return continue sending serial string
    {
        uart_sendchar(tx_string[i++]); //call to uart character sending function
    }
    uart_sendchar('\r');              //loop through
sequence of i until finished
}

```

```

int uart_getstring(char rx_string[])
{
    int i = 0;
    char c;
    c = uart_getchar();
    while (c != '\r') //0x3A)
    {
        rx_string[i++] = c;
        c = uart_getchar();//USART_GetChar(&USART);
    }
    return i;
}

```

int string2int(char* str) // e.g. str="356\0". Then value=356.

```

{
    int value = 0;
    int i, j, zeroNum, tempVal, signFlag;
    int strl = strlen(str);

    signFlag = 0;
    i = 0;
    if (str[0]=='-') // a negative number
    {
        signFlag = 1;
        i = 1;
    }

    for(j<strl; i++)
    {
        tempVal = str[i] - 0x30;
        zeroNum = strl - i - 1;
        for(j=0;j<zeroNum;j++)
        {
            tempVal *= 10;
        }
        value += tempVal;
    }

    if (signFlag)
        value = -value;

    return value;
}

```

char* int2string(int value) // e.g. value = -356. Then str = "-356\0".

```

{
    char* str;
    int i,j;
    int zeroNum, decBit, signFlag;
    int divider = 10;

```

```

if (value<0)
{
    signFlag = 1;
    value = -value;
}
else
    signFlag = 0;

decBit = 1;
while((value/divider)>0)
{
    decBit++;
    divider *= 10;
}

str = (char*)malloc((decBit+signFlag+1) * sizeof(char));
i = 0;
if (signFlag)
{
    str[i++] = '-';
}

divider /= 10;
zeroNum = decBit;
while (i<(decBit+signFlag) && value!=0)
{
    str[i++] = (value/divider) + 0x30;
    value -= (value/divider)*divider;
    divider /= 10;
    zeroNum--;
}

if (zeroNum)
{
    for (j=0; j<zeroNum; j++)
        str[i++] = '0';
}

str[i] = '\0';

return str;
}

```

```

int panservo(int pos)
// change the pan-servo position by pos
// if pos==255, then set the pan-servo to the central position
// return current position
{
    int curPos;
    char str1[4];
    char *str2, *str3;
    int str1;

```

```

if (pos == 255)
{
    uart_sendstring("sv 0 128\r"); // set pan-servo to the central position
    curPos = 128;
}
else
{
    uart_sendstring("gs 0\r");
    str1 = uart_getstring(str1); // ACK
    str1 = uart_getstring(str1); // '\0' is included

    curPos = string2int(str1);
    curPos += pos;

    if (curPos < 46)
        curPos = 46;
    else if (curPos > 210)
        curPos = 210;

    str2 = int2string(curPos);
    str1 = strlen(str2);

    str3 = (char*)malloc((5+str1+1)*sizeof(char));
    strcpy(str3, "sv 0 \0");
    str3 = strcat(str3, str2); // now, e.g. str3 = "sv 0 curPos\0"
    str3[strlen(str3)] = '\r';

    uart_sendstring(str3);
    str1 = uart_getstring(str1); // ACK
}

return curPos;
}

```

```

int tilt servo(int pos)
// change the tilt-servo position by pos
// if pos==255, then set the tilt-servo to the central position
// return current position
{
    int curPos;
    char str1[4];
    char *str2, *str3;
    int strl;

    if (pos == 255)
    {
        uart_sendstring("sv 1 128\r"); // set pan-servo to the central position
        curPos = 128;
    }
    else

```

```

    {
        uart_sendstring("gs 1\r");
        str1 = uart_getstring(str1); // ACK
        str1 = uart_getstring(str1); // '\0' is included

        curPos = string2int(str1);
        curPos += pos;

        if (curPos < 46)
            curPos = 46;
        else if (curPos > 210)
            curPos = 210;

        str2 = int2string(curPos);
        str1 = strlen(str2);

        str3 = (char*)malloc((5+str1+1)*sizeof(char));
        strcpy(str3, "sv 1 \0");
        str3 = strcat(str3, str2); // now, e.g. str3 = "sv 0 curPos\0"
        str3[strlen(str3)] = '\r';

        uart_sendstring(str3);
        str1 = uart_getstring(str1); // ACK
    }

    return curPos;
}

TPACKET tpacketDecode(char* tstr)
{
    //lcdInit();lcdGoto(1,0);lcdString(tstr);
    TPACKET tpacket;

    int count1= 2;
    int count2 = 0;
    int temp[8] = {0};
    //temp[0]=temp[1]=temp[2]=temp[3]=temp[4]=temp[5]=temp[6]=temp[7]=0;

    int length = strlen(tstr);

    for(int i=0;i<8;i++)
    {
        char* buff;
        count2=count1;
        while((tstr[count2]!=' ')&&(count2<length))
        {
            //lcdGoto(1,0);
            // lcdInt(count2);
            count2++;
        }

        if((count2-count1)==1)
            temp[i] = atoi(tstr[count1]);
    }
}

```

```

else if((count2-count1)==2)
{
//      lcdInit();lcdGoto(0,0);lcdString("Here");lcdGoto(1,0);lcdInt(i);delay_ms(500);
      buff[0] = tstr[count1];
      buff[1] = tstr[count1+1];
      buff[2] = '\0';
      //temp[i] = atoi(tstr[count1])*10 + atoi(tstr[count1+1]);
      temp[i] = atoi(buff);
      /*lcdInit();
      lcdGoto(0,0);
      lcdInt(temp[i]);
      lcdGoto(1,0);
      lcdString(buff);
      delay_ms(500);*/

}
else if((count2-count1)==3)
{
      buff[0] = tstr[count1];
      buff[1] = tstr[count1+1];
      buff[2] = tstr[count1+2];
      temp[i] = atoi(buff);
      //temp[i] = temp[i] = atoi(tstr[count1])*100 + atoi(tstr[count1+1])*10 +
      atoi(tstr[count1+2]);//atoi(buff);
      /*
      lcdInit();lcdGoto(0,0);lcdString(tstr);
      lcdGoto(1,0);lcdString(buff);
      lcdGoto(1,4);lcdInt(temp[i]);
      lcdGoto(1,8); lcdInt(count1);lcdGoto(1,11);lcdInt(count2);
      delay_ms(3000);*/

//      lcdGoto(1,6);
//      lcdString(buff);

}
//      lcdGoto(0,0);
//      lcdInt(count1);
//      lcdGoto(0,5);
//      lcdInt(count2);
//      lcdGoto(1,0);
//      lcdInt(temp[i]);
//      delay_ms(2500);
      count1 = count2+1;

}

tpacket.mx = temp[0];
tpacket.my = temp[1];
tpacket.x1 = temp[2];
tpacket.y1 = temp[3];
tpacket.x2 = temp[4];
tpacket.y2 = temp[5];
tpacket.pixels = temp[6];
tpacket.confidence = temp[7];

```

```

/*  int i,k,strleft,strtight,numlen;
    char *str;
    int ll = strlen(tstr);
    int temp[8];

    k = 0;
    strleft = 0;
    while(tstr[strleft]!=' ')
        strleft++;

    i = strleft;
    strtight = strleft;
    while(i<ll)
    {
        strleft = strtight;
        strtight++;
        i++;
        while(tstr[strtight]!=' ' && tstr[strtight]!='\0')
        {
            strtight++;
            i++;
        }
        numlen = strtight - strleft - 1;
        str = (char*)malloc((numlen+1)*sizeof(char));
        strncpy(str, tstr+strleft+1, numlen);
        str[numlen] = '\0';

        temp[k++] = string2int(str);
        free(str);
    }

    tpacket.mx = temp[0];
    tpacket.my = temp[1];
    tpacket.x1 = temp[2];
    tpacket.y1 = temp[3];
    tpacket.x2 = temp[4];
    tpacket.y2 = temp[5];
    tpacket.pixels = temp[6];
    tpacket.confidence = temp[7];*/

    return tpacket;
}

```

```

TPACKET tpacketDecode1(char* tstr, char number_bytes)
{lcdInit();

```

```

TPACKET tpacket;
char tstr_byte = 0;

```

```

char count = 0; // keeps count of bytes (1 to number_bytes)

```

```

char count_num = 0; //keeps count of digits in a number (1 or 2 or 3)
char count_temp = 0; // keep count of the number of the data being stored in temp.

char temp[8] = {0};
char num_array[3] = {0};
char num = 0;

count_num = 0;

for (count = 2; count < number_bytes + 1 ; count++ )
{
if (count == number_bytes)
tstr_byte = ' ';
else
tstr_byte = tstr[count];

if (tstr_byte == ' ')
{ // comes here when byte recieved is a space = 0x20
//lcdInt(count);
if (count_num == 1)
num = (num_array[0] - 0x30);
if (count_num == 2)
num = ((num_array[0] - 0x30) * 10) + (num_array[1] - 0x30);
if (count_num == 3)
num = ((num_array[0] - 0x30) * 100) + ((num_array[1] - 0x30) * 10) + (num_array[2] - 0x30);

count_num = 0;
temp[count_temp++] = num;

num = 0;
for (count_num = 0; count_num < 3; count_num++)
{
num_array[count_num] = 0;
}
count_num = 0;
//while(1);

}
else
{
num_array[count_num++] = tstr_byte;

}

}

tpacket.mx = temp[0];
tpacket.my = temp[1];

```

```

tpacket.x1 = temp[2];
tpacket.y1 = temp[3];
tpacket.x2 = temp[4];
tpacket.y2 = temp[5];
tpacket.pixels = temp[6];
tpacket.confidence = temp[7];

return tpacket;

}

```

```

SPACKET spacketDecode(char* tstr)
{
    SPACKET spacket;
    int i,k,strleft,strtight,numlen;
    char *str;
    int ll = strlen(tstr);
    int temp[6];

    k = 0;
    strleft = 0;
    while(tstr[strleft]!=' ')
        strleft++;

    i = strleft;
    strtight = strleft;
    while(i<ll)
    {
        strleft = strtight;
        strtight++;
        i++;
        while(tstr[strtight]!=' ' && tstr[strtight]!='\0')
        {
            strtight++;
            i++;
        }
        numlen = strtight - strleft - 1;
        str = (char*)malloc((numlen+1)*sizeof(char));
        strncpy(str, tstr+strleft+1, numlen);
        str[numlen] = '\0';

        temp[k++] = string2int(str);
        free(str);
    }

    spacket.r = temp[0];
    spacket.g = temp[1];

```

```

    spacket.b = temp[2];
    spacket.rSigma = temp[3];
    spacket.rSigma = temp[4];
    spacket.rSigma = temp[5];

    return spacket;
}

SPACKET getMean(int panPos, int tiltPos, char *CameraRx)
{
    //char CameraRx[40];
    SPACKET spacket;

    ServoC0(panPos);
    ServoC1(tiltPos);
    delay_ms(600);
    uart_sendstring("rf\r");
    uart_getstring(CameraRx);
    delay_ms(200);
    uart_sendstring("rf\r");
    uart_getstring(CameraRx);
    delay_ms(200);
    uart_sendstring("gm\r");
    uart_getstring(CameraRx); // ACK
    uart_getstring(CameraRx);
    spacket = spacketDecode(CameraRx);

    return spacket;
}

```

REFERENCES

CMUcam Manual v2.00 for the CMUcam v1.12 firmware (<http://www.cs.cmu.edu/~cmucam>)