A Quadruped Walking Robot

Frank Bergschneider
Robot Name: Mini-Dog
EEL 5666 Intelligent Machine Design Laboratory

Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz
TAs: Ryan Chilton and Josh Weaver

Table of Contents

1. Abstract 3
2. Executive Summary 3
3. Introduction 3
4. Integrated System 4
5. Mobile Platform 5
6. Actuation 7
7. Sensors 8
8. Behaviors 11
9. Experimental Layout and Results 13
10. Conclusion 16
11. Documentation 16
12. Appendices 17

1. Abstract

Mini-Dog is an intelligent autonomous quadruped robot designed to follow a designated leader using color tracking with a camera. Legged robots excel at traversing rugged terrain and open new possibilities for troop support roles, autonomous exploration, and the study of walking mechanics. The Mini-Dog is a proof of concept robot that demonstrates quadruped locomotion, which could be further developed to incorporate autonomous navigation by GPS and scaled up to carry supply loads.

2. Executive Summary

Mini-Dog's goal is to demonstrate quadruped locomotion and follow a designated leader by using color tracking. The leader will hold a colored piece of poster board and Mini Dog will track the leader as they move around. The robot consists of a mobile platform with four legs and a Droid phone with an IP Webcam application installed, and a laptop for image processing. The mobile platform consists of a microcontroller board that handles all low level servo control, sensor inputs, LED direction display, and also direction arbitration based on input from the sensors and direction information received from the laptop. The color tracking algorithm is performed on a remote laptop and direction data is sent to the robot by an Xbee connection. The color tracking program contains logic for determining if the robot is too close to the leader and if the color is present or if simply color noise from the environment is present. When the robot is turned on, it moves to a static standing position and waits for direction information from the color tracking program. If no direction information is received after approximately 1 minute, then the robot enters search mode and begins to search for the leader. Once the leader is found, the robot will actively follow the leader until it reaches a close position and waits for the leader to continue moving.

3. Introduction

Legged robots have reached the point where they are transitioning from the areas of research and development to commercially available products. Recent developments include Esko Bionics Exoskeleton, IHMC Mina Exoskeleton and Boston Dynamics' LS3 Big Dog, Little Dog and PETman robots. The focus of this project is on quadruped walking robotics. The Mini-Dog is designed to be an autonomous quadruped walking robot capable of following a leader using a camera. It will employ quadruped locomotion to follow its target. The robot is specifically inspired by Boston Dynamics' LS3 Big Dog and Little Dog quadruped robots.

The robot will follow a leader using a camera to track a specifically colored target worn by the leader. The camera is a Droid cell phone with an IP camera application installed, and OpenCV/C++ is used for image processing on an off board laptop computer. The laptop will handle the image processing and communicate to the robot which direction to go. The robot will use this information and do trajectory planning onboard. The laptop and robot communicate wirelessly using Xbee radio modules. It will employ 1 sonar sensor and 2 IR sensors for object detection and avoidance. The sonar would be used as an input into the trajectory planning of the robot.

4. Integrated System

The Mini Dog robot consists of two parts: the mobile robotic platform and a stationary laptop computer. The mobile robotic platform consists of a Xmega microcontroller board, Xbee Series 1 radio module, sonar and IR sensors, LED array, Motorola Droid with IPWebcam application installed, a 2200 mAh LiPo battery, and 4 legs actuated by 2 servos on each leg. The laptop computer consists of an Xbee USB Explorer module with an Xbee Series 1 module installed, Visual Studios 10 with the OpenCV libraries, ManyCams Virtual Webcam software, and IP Webcam Adapter software running on Windows 7 OS. A Linksys WRT54GL Wireless Broadband router is used for an IP/TCP connection between the Droid phone and the laptop.

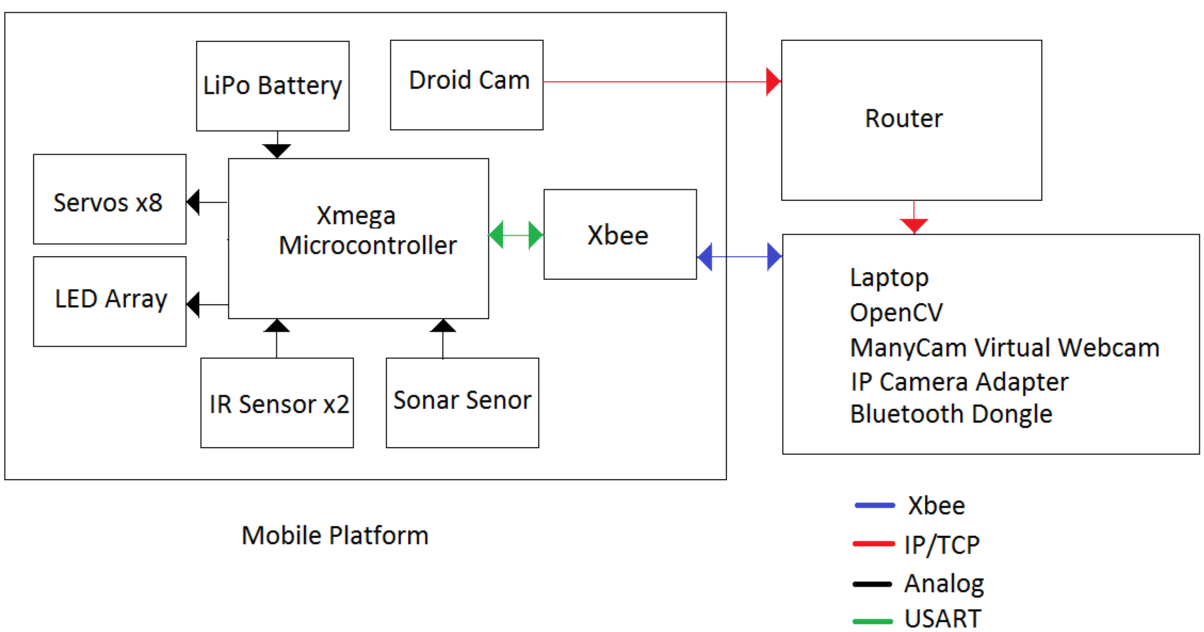

Figure 1: System Block Diagram

The microcontroller board is the Epiphany ATxmega128A1U from OOTB Robotics. The development environment used for the microcontroller is Atmel Studio and the microcontroller is programmed in the language is C. The microcontroller programming software is chip45boot2 programmer. The color tracking camera software was developed in Visual Studios 10 with the OpenCV libraries and the language is C++. The below tables contain a summary of the primary hardware and software components used for the robot.

Component	Location	Function	Description
Motorola Droid Phone	Robot	IP Webcam	Android v2.2.3, Arm Cortex A8 550 MHz Processor, 5MP camera
Xmega Board	Robot	Controller	ATxmega128A1U Development Board
Xbee Series 1 Module	Robot/Laptop	Xbee Connection	Digit International Series 1 Xbee Radio
Xbee Explorer	Laptop	Xbee Connection	Funspark Xbee USB Explorer
Laptop	Laptop	Image Processing	Gateway MS2274 with Windows 7 OS
Router	Laptop	IP Connection	Linksys WRT54GL Wireless Broadband

Table 1: Primary Hardware Components

Software	Location	Function	Description
IP Webcam App	Droid Phone	IP Webcam	Video over IP
IP Camera Adapter	Laptop	IP Webcam	Network camera adapter
ManyCam Virtual Webcam	Laptop	IP Webcam	Mount IP Webcam as virtual webcam
MS Studio 10	Laptop	Image Processing	Color tracking from IP Webcam video stream
X-CTU	Laptop	Xbee Setup	Configure Xbee network addresses

Table 2: Primary Software Components

5. Mobile Platform

The mobile platform has evolved from the initial design to a shorter, more stable design. The knee servos have also been mounted on the outside of the thigh to avoid hitting the body, and to make the thigh as shorter as possible to minimize the torque on the hip servos. The body was constructed from ½” aluminum angle brackets because of the high torques and loads the body experiences during walking. The legs were constructed from 1/8” laminated plywood. This same plywood was used as top cover for the body to mount the sensors and Droid phone. A 1/16” steel hurricane strap was bent into a mount to hold the Droid phone. Small angles brackets were used to mount the sensors to the top covers. Primarily, 4-40 and 6-32 screws and lock nuts were used to fasten components together. Rubber doorstops were added to the bottom of the legs to provide more traction between the floor and legs, which improved the walking behavior.

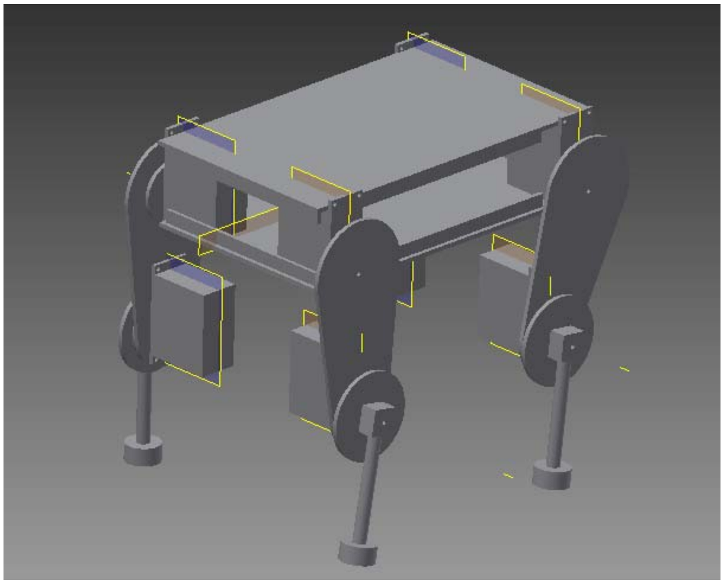

Figure 2. Initial Mobile Platform Design

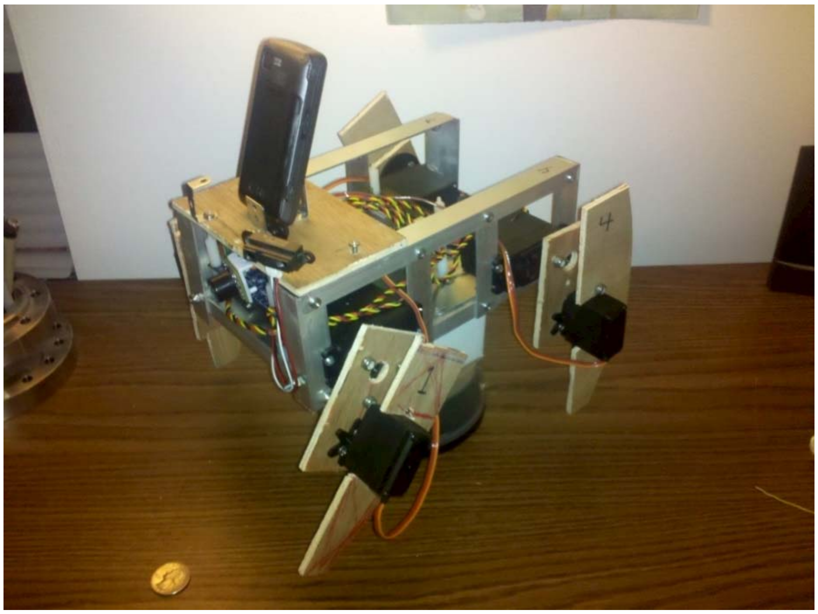

Figure 3. Assembled Mobile Platform

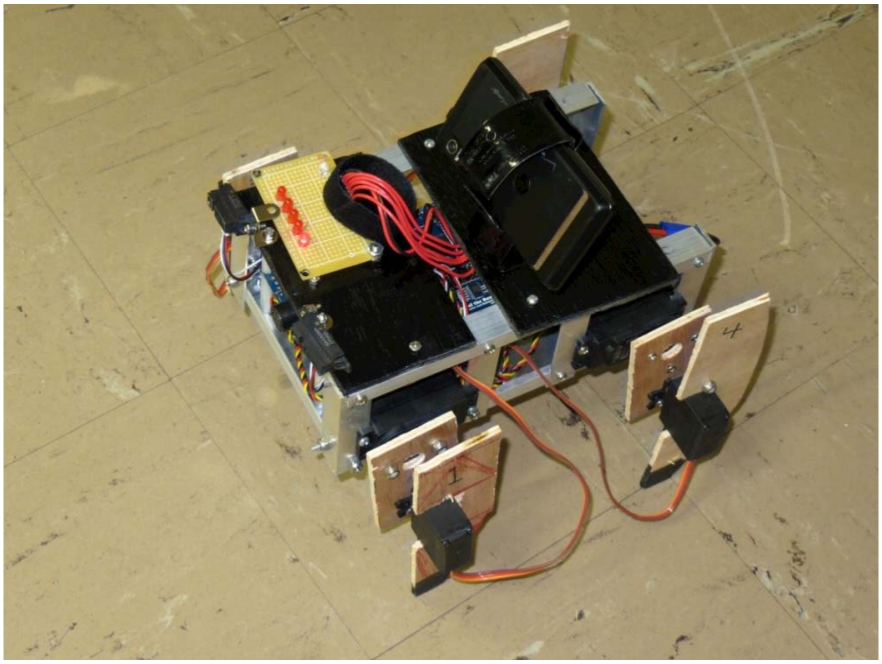

Figure 4. Final Mobile Platform

6. Actuation

The robot will move using 4 actuated legs employing a 4-legged animal gait (e.g. Similar to a cat or dog, etc) with servo actuated hip and knee joints with a passive shock absorber/spring shank segment. It will employ 4 Hitec HS-755Hb 1/4 Scale Servo HRC33755S for the robot's hip, and 4 HD Power servos for the knees. Below is a sample gait pattern for one leg. All four legs implement this walking pattern simultaneously for forward and turning locomotion. The pattern is implemented in parts for each leg during a single gait cycle. For an example of the forward movement sequence, the front left leg pulls back and extends, then the front right and back leg shift backwards pushing the body forward, the front left then shifts back, the back right swings back and extends forward, the front right and back left then shift back, and finally the back right shifts back to complete the cycle. Because of concerns about dynamic stability and excessive load on the servos, the walking pattern is executed in complete cycles even if any object is detected by the sensors. Because each gait pattern takes approximately 1 second to execute, this leads to decreased responsiveness when obstacle avoidance behaviors are necessary. For turning, the basic change in the gait pattern is to decrease the joint angles on the desired turning side, and increase the joint angles on the opposite. For a hard turn, this same pattern was applied with the addition of keeping the back leg on the turning side stationary to decrease forward movement.

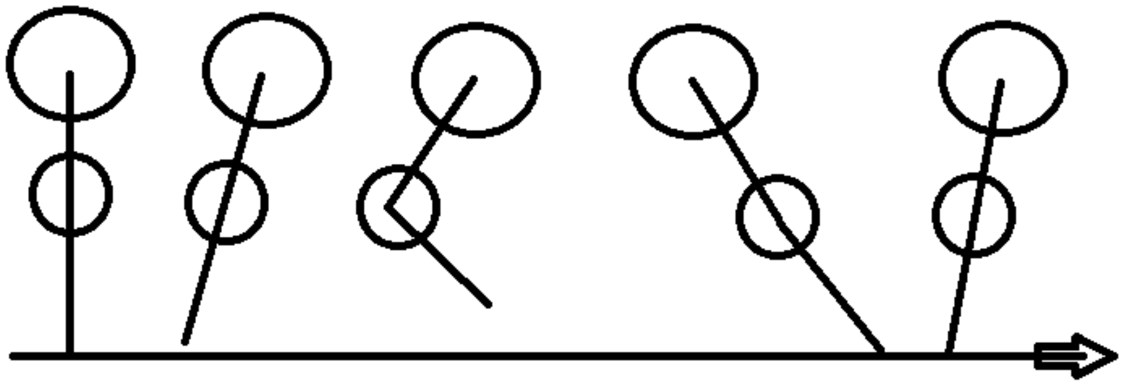

Figure 5: Walking Gait Pattern

7. Sensors

The sensors used for the robot are: (x2) IR Distance sensor, (x1) Sonar Range Finder, and Motorola Droid IP Camera with the color tracking vision system. The below table list the sensors used, and each sensor is detailed below.

Sensor	Function	Qty
LV-MaxSonar-EZ4	Forward Object Sonar Range Finding	1
Sharp GP2Y0A21	Side IR Distance Sensing	2
Droid Camera	Image Acquisition	1

Table 3: Sensor Selection

Sonar Sensor

The LV-MaxSonar-EZ4 is a sonar range finder with a working range of 1-21". Currently, the analog output is being sampled, which gives a scaled voltage from 0-Vcc (3.3 V). The output voltage is extremely noise, which is due to the sensor's electronic noise susceptibility. A lowpass filter was implemented in software for this sensor. The filter takes a weighted average of current and two previous samples. The equation is given by

$$\text{Filtered } x[n]=(4/6)*x[n]+(1/6)*x[n-1]*(1/6)*x[n-2]$$

LV-MaxSonar®-EZ4™ Circuit

The LV-MaxSonar®-EZ4™ sensor functions using active components consisting of an LM324, a diode array, a PIC16F676, together with a variety of passive components.

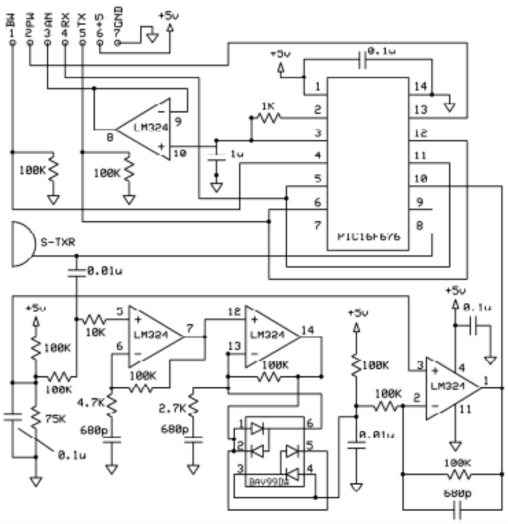

Figure 6: Sonar Circuit

IR Distance Sensor

The Sharp GP2Y0A21 is a IR Distance sensor with a working range of 4-32”. It outputs a voltage scaled from 0-Vcc (3.3 V). A lowpass filter was implemented in software for these sensors.

■Absolute Maximum Ratings (T_a=25℃,Vcc=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	-0.3 to +7	V
Output terminal voltage	V_O	-0.3 to V_{CC}+0.3	V
Operating temperature	T_{opr}	-10 to +60	℃
Storage temperature	T_{stg}	-40 to +70	℃

Figure 7: IR Specification

Color Tracking Vision System

The overall objective is to follow a leader using quadruped locomotion and a color tracking vision system. The color tracking system is the “special” system, which allows to the robot to follow a leader. The system consists of a Motorola Droid with Android OS 2.3.2 and IP Webcam Application installed with a Gateway MS2254 laptop running Visual Studios for image processing. The Droid IP Webcam required two extra programs to access Droid cam MJPEG stream on the laptop: IP Webcam Adapter and ManyCam Virtual Webcam. The virtual webcam was not displayed in the Device Manager in Windows, but was listed as WebCam 0 when accessed by OpenCV. There is also approximately a 1-2 second lag from Droid camera to processed video. The IP Webcam app was written by Pavel Khlebovich.

The image processing algorithm performs a color thresholding of the desired color, calculates the number pixels of the desired color and the center of mass position of the pixels. This information is used to determine if the number of pixels is greater than the minimum number of pixels required to detect the colored badge. This is to prevent noise pixels from being counted as the badge. If the number of pixels is greater than the minimum, then the x position of the CoM is sent to the robot. One problem encountered was that the images from the IP Webcam were very noisy and delayed at least 1 second. To counter the effects of the noise, the OpenCV morphological operation erode and dilate were used to shrink small patches of noisy pixels and grow large patches of closely clustered pixels respectively. The delay seemed to be unavoidable due to the chain of programs the video was being passed through.

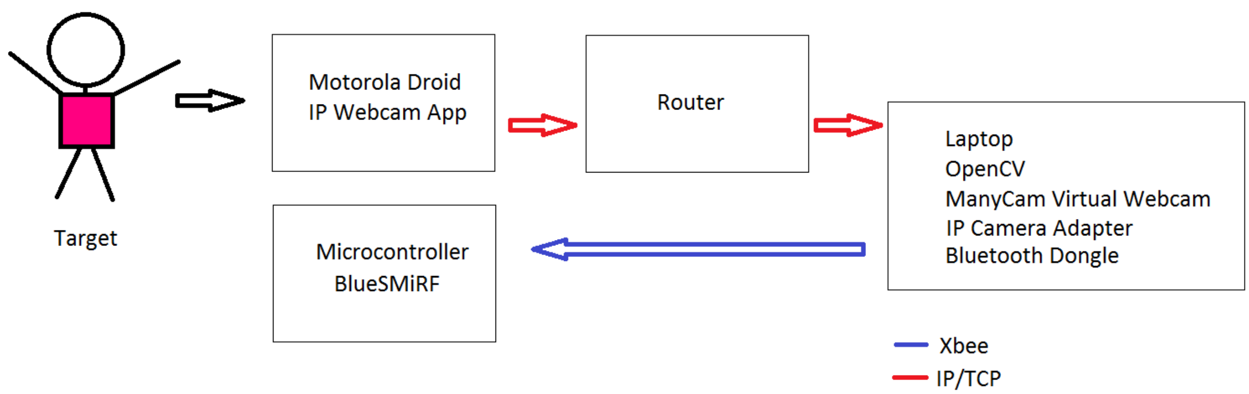

Figure 8: Color Tracking Vision System Block Diagram

The HSV color scale was used to characterize the colored badge that the robot will follow. Neon green was selected based on the results of experiments with the vision system. The HSV values for neon green are approximately 48-53; 200-230; 225-250. After testing with neon green, it was determined that hot pink resulted in less noisy and stood out from the environment better. The target was switched to hot pink with HSV values of approximately 330; 100; 100.

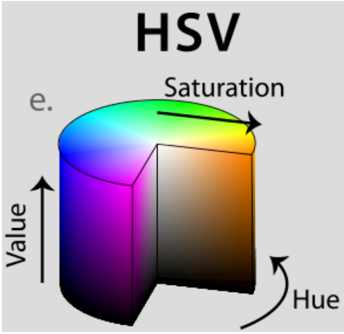

Figure 9: HSV Color Classification

Communication

The final communication module was used the Xbee 1mW Trace Antenna Series 1 radio modules. The Xbees provided a wireless communication link between the laptop and the robot. A SparkFun Xbee USB Explorer was to interface the Xbee module to the USB port on the laptop. The Xbee modules were configured using X-CTU from Digi International. The PAN IDs were changed to 3000 to prevent communicating on other channels given the number of other robots using Xbee.

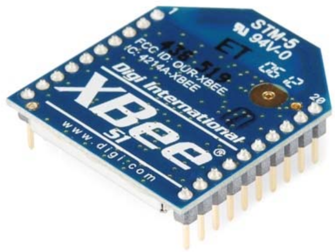

Figure 10: Xbee Module

8. Behaviors

Mini-Dog has two basic behaviors: Follow the leader and search for the leader. Once the robot is turned on, it moves all of its servos to the zero positions and waits for approximately 5 seconds. The zero position is 90 degrees for all the servos. Once it has zeroed the servos, the robot waits for approximately 1 minute to get direction information from the laptop. During this period, the robot doesn’t move. If no directions are received in this time period, then the robot starting walking in search of the leader. During this period, the robot relies on sensor input only and tries to move forward in a direction without any obstacles. Because of the difficulties with turn, the threshold for the robot to perform a turn in this mode are very low. Movement in this mode is essentially random because of the interactions with the environment. The robot will continue to explorer for about 2 minutes, and then return to a stationary state. The robot will return to search mode after approximately 1 more, and this cycle will continue until a leader is found. Obstacle avoid is based on upper and lower sensor threshold for the IR and sonar sensors. There is also logic for sudden large increases in the sensor values which indicate sudden movement near the robot.

If the leader's color is detected, then the robot receives a direction from the laptop to go forward, turn left or turn right to follow the leader. As long as the leader is found, the robot will follow indefinitely. If the leader's color is found but the number of pixels of the color is too large than the robot does not move until the number of pixels decreases below a threshold. This prevents the robot from running into the leader. The flow charts for the robot and color tracking program are given below.

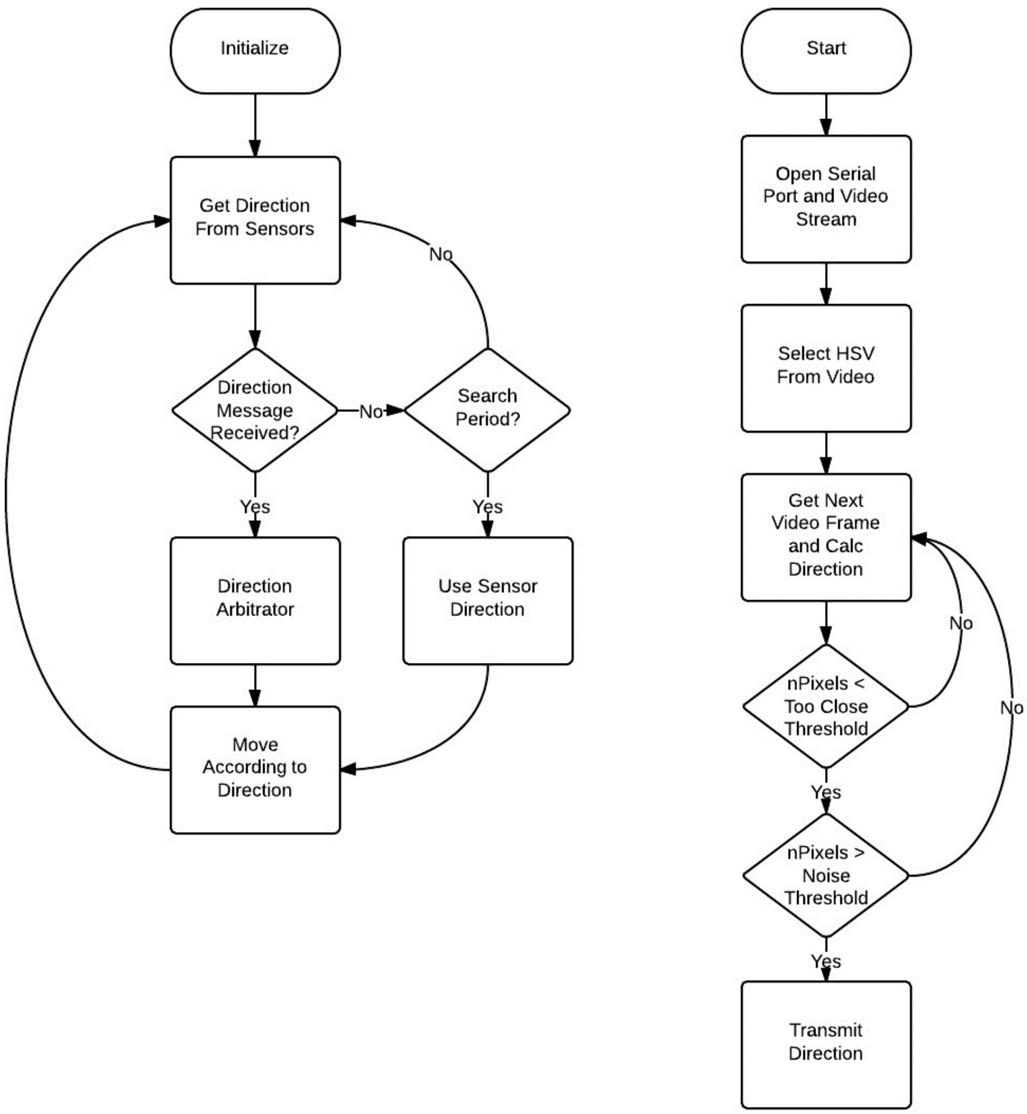

Figure 11: Mini Dog Flow Chart

9. Experimental Layout and Results

Walking Algorithm Experiments

These experiments are ongoing. I have finished assembling the legs, and I have been experimenting with different walking patterns. I have moved away from a stiff cautious gait to one that is more biologically inspired by a cat's gait. I have found that to be able to smoothly move the leg forward, it must be rotated back and then lift the shin to avoid scrapping the foot on the ground.

Droid Camera Image Processing Experiment

A small colored target was placed in different environments and lighting conditions to test the effects on the color tracking system. Several colors were tested and neon green was chosen because it was very dissimilar to any colors found in the environment. The vision system would detect large amounts of noise if the selected color was very similar to other colors in the image, eg if neon pink was chosen, noise pixel would appear occasionally in any red or orange in the image. The color tracking program on the laptop will be calibrated during operation by selecting the color manually in the operating environment (mouse click on the desired color).

Neon Green HSV	48-53 ; 200-230; 225-250
Pixel Threshold for Target Presence	1000

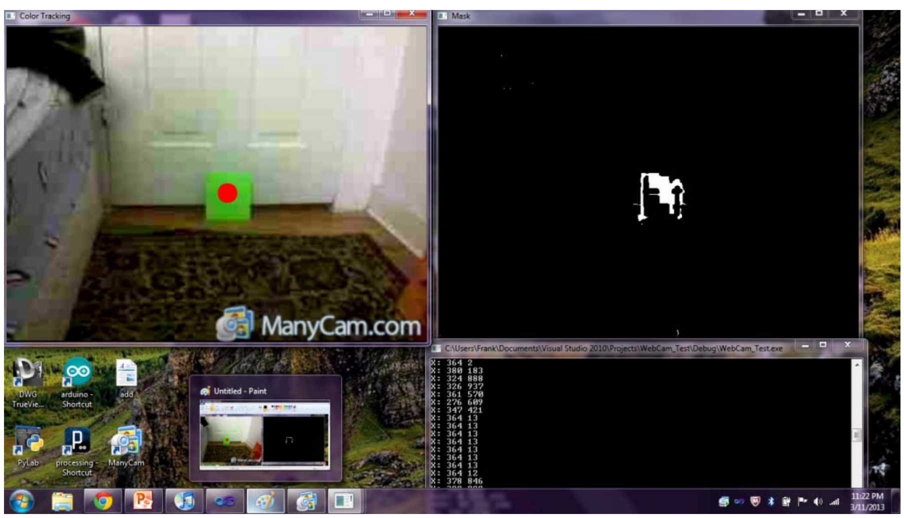

Figure 12: Processed Droid camera stream

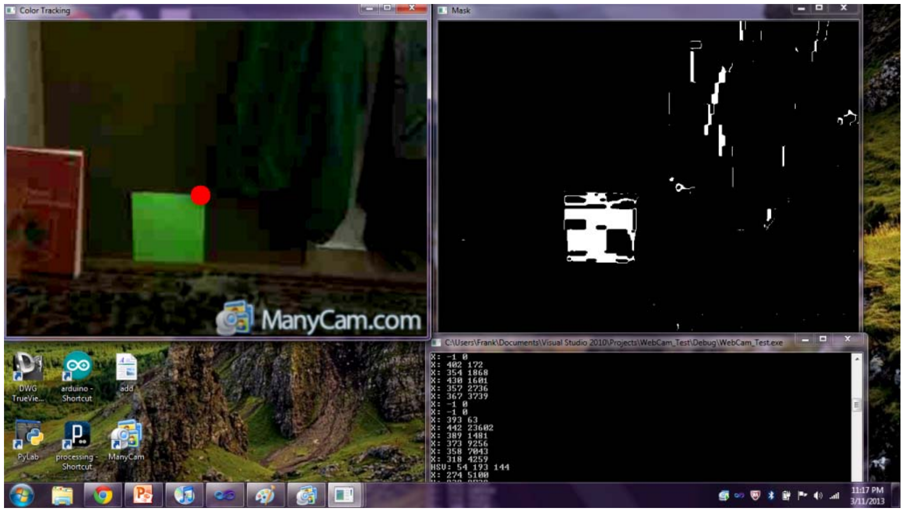

Figure 13: Processed Droid camera stream with noise from environment

Sonar Sensor Characterization

The sonar was mounted on the robot, and the sensor values were recorded as a large board was moved toward the sensor. The analog output of the Sharp sonar sensor is known to be very susceptible to electronic noise. The first plot shows considerable variation in output voltage for a slowly moving target. This led to using a low pass filter in software. The second plot shows a smoother (noisy samples were manually removed) curve for reference.

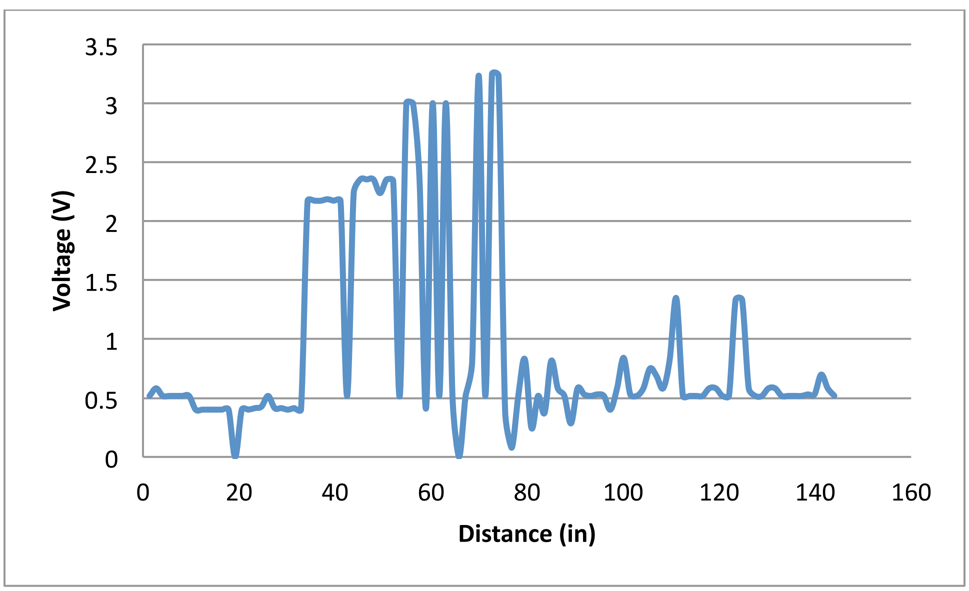

Figure 14: Noisy Analog Output from LV-MaxSonar-EZ4

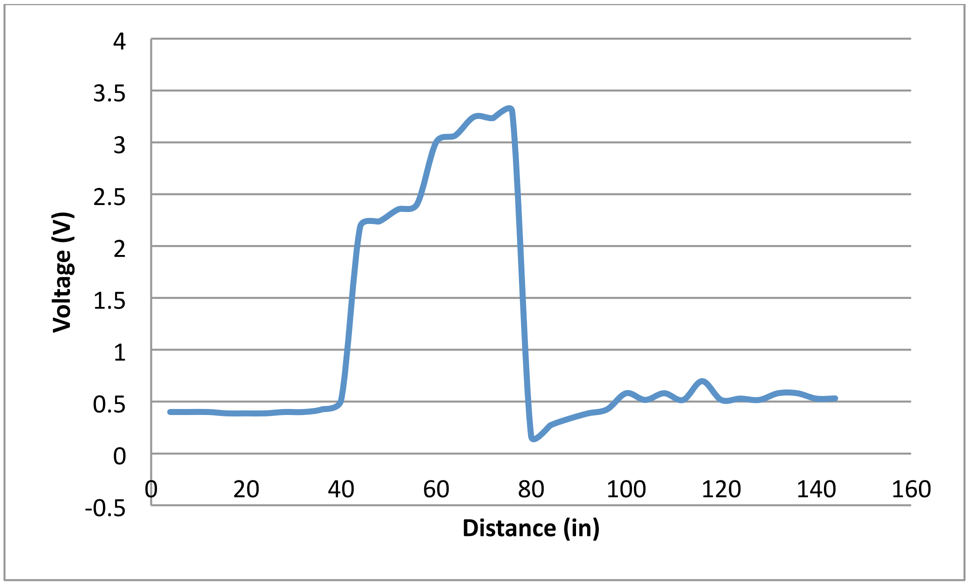

Figure 15: Reference Analog Output from LV-MaxSonar-EZ4

IR Sensor Characterization

The IR sensors were mounted on the robot, and the sensor values were recorded as a large board was moved toward the sensors. The analog output of the sensor was much more consistent than the sonar sensor. There were slightly differences between the two sensors. One was show more susceptibility to noise. This may be due to faulty wiring on the noisier sensor. Lowpass filters will also be implemented in software for these sensors.

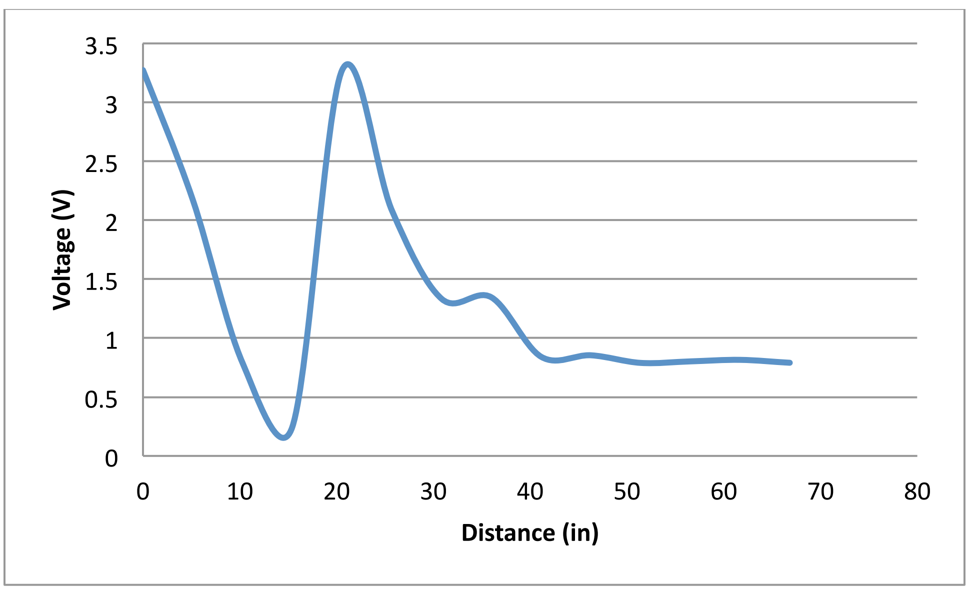

Figure 16: Analog Output from Sharp GP2Y0A21 Distance Sensor

10. Conclusion

The Mini Dog robot has taught me some valuable lessons about embedded programming, image processing, and commercially valuable sensors and components. I learned that simplicity is a very good thing. The complexity of the walking algorithm took a lot of time to develop and I wish I had got to focus more on the Xmega and OpenCV aspects of the project. I will definitely continue developing the robot. There are quite a few changes I would like to make in the future to the robot. I would like to add a 3rd degree of freedom to each leg to enhance the turning abilities of Mini Dog. This would allow for much sharper turns with less overall servo movements. It would also good to have foot switches mounted on the bottom of the foot to indicate if the foot is touching the ground. The foot switches would be used as feedback for gait control. Ideally, the robot would be able to walk over uneven ground, and foot down information would be valuable. I would also like to have an embedded camera with on board image processing via an Odroid or BeagleBoard. Wireless communication proved to be problematic and having a self-contained robot would make it more reliable and practical.

11. Documentation

Mini-Dog Website	https://sites.google.com/site/minidogfbb/
Epiphany Xmega Board	http://ootbrobotics.com/
LV-MaxSonar-EZ4 Sonar:	http://www.pololu.com/file/0J85/gp2y0a21yk0f.pdf
Sharp GP2Y0A21 IR Sensor:	http://www.jameco.com/Jameco/Products/ProdDS/2157335.pdf
Xbee Datasheet:	https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf
HSV	http://en.wikipedia.org/wiki/File:Hsl-hsv_models.svg
HD MG1501 Servos	http://www.pololu.com/catalog/product/1057
GForce 2200mAh LiPO	http://www.valuehobby.com/
ManyCams	http://www.manycam.com/
IP Webcam	https://play.google.com/store/apps/details?id=com.pas.webcam&hl=en
IP Camera Adapter	http://ip-webcam.appspot.com/

12. Appendices

A. Mini Dog Code

Adapted from OOTB Robotics Library

```
/*
* Mini Dog
 *
 * Major Change log:
 * 3/27/13: Updated direction arbitrator logic
 * 3/16/13: Updated walking algo
 * 3/7/13: Updated walking algo
 *
 *  Created: 1/20/2013
 *  Author: Frank Bergschneider
 */

//#include <ctype.h>
//#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <string.h>

#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "ADC_manual.h"


#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))
#define analogChannel1 0  //analog channel on port A ADC Channel 0
#define analogChannel2 1  //analog channel on port A ADC Channel 1
#define leftIRPin 3
#define rightIRPin 2
#define sonarPin 1
#define leftIRThr_U 200
#define leftIRThr_L 60 //40
#define rightIRThr_U 200 //90
#define rightIRThr_L 60
#define sonarThr_U 254
#define sonarThr_L 50
#define sonarThr_U2 110
#define sonarThr_L2 60
#define BT_str usartD0_str//usartD0_str //Bluetooth port, Pin2=Rx, Pin3=Tx

uint8_t lastLR;             //last side direction detected used for hard turns
uint8_t direction;
uint8_t direction1;
uint8_t direction2;
uint8_t direction3;
uint8_t directionFromCamera;
uint8_t objectFound;
```

```
uint8_t dT=10, dK=15;
uint8_t sZ=90; //all servos are zeroed at 90 deg
uint8_t countSinceLastMsg;
//uint8_t msgCount;
uint8_t sonar_Filtered;
uint8_t rightIR_Filtered;
uint8_t leftIR_Filtered;
uint8_t senseturn;

void talkToLaptop(void); //talk to laptop vision system
void directionMessage(void);      // calls talkToLaptop three times to get direction
void turnLeft(void);
void turnRight(void);
void goForward(void);
void turnLeftHard(void);
void turnRightHard(void);
void nextState(void);
void zeroServos(void);
void cylon(void);
void cylon_slow(void);

int main(void)
{

       uint8_t move=0;

       //global var initializations
       senseturn=0;
       sonar_Filtered=50;
       rightIR_Filtered=50;
       leftIR_Filtered=50;
       direction=1; direction1=1; //direction2=5; direction3=5;
       directionFromCamera=1;
       objectFound=0;
       countSinceLastMsg=0;
       //msgCount=0;
       lastLR=4;
       int go=1, BTLED=0;

       clockInit();
       ATtinyServoInit();
       PORTE.PIN3CTRL =  PORT_SRLEN_bm; //decrease slew rate on Tiny pins

       //Leg angles names: side|Thigh or Knee|Zero or Forward or Back
       uint8_t sZ=90; //all servos are zeroed at 90 deg


       /* ISR complete later
       //initialize 5 Hz timer for sensing
       //TCF0.CTRLA=TC_CLKSEL_DIV64_gc; //0x01 select system clk /64
       TCF0.CTRLA=TC_CLKSEL_DIV1_gc;
       TCF0.PER=10000*64; //20 ms period
       TCF0.CNT=0; //set counter to 0
       //TCF0.INTCTRLA=TC_Ckl;
       */
              //PORTE.DIRSET=0x80;//xbee pin
              PORTE.OUTSET=PIN7_bm; //?
              PORTE.DIRSET=PIN7_bm; //set PD3 as Tx/out
```

```
        PORTE.DIRCLR=PIN6_bm; //Set D2 as Rx/in


usartInit(&USARTC0,115200);//USB port
usartInit(&USARTD0,9600);//Bluetooth port
usartInit(&USARTE1,9600);
sei(); //global interrupt enable

//PORTF.OUT = 0x00;
fprintf(&USB_str,"Start\r \n");
fprintf(&Xbee_str,"Start Xbee\r \n");

//Blink Debug LED and blue LED
PORTR.DIRSET=0x02;
PORTB.DIRSET=0xFF;
PORTB.OUT = 0xFF;
PORTD.DIRSET=0xFF;
PORTD.OUT = 0x20;
zeroServos();
//cylon_slow();
PORTR.OUT = 0xff;  //Inverted logic on port r!
//cylon_slow();
PORTB.OUT = 0x00;

//ADC likes being here, needs time for clock to initialize?
ADC_init();

while(go){
        //Move laptop comm to ISR at 2 Hz
        talkToLaptop();
        //directionMessage();
        nextState();

        //R side servos: 2,3  L side servos: 1,4
        if (direction==5){
                //Straight
                PORTD.OUT=0x20;
                if (move){
                goForward();
                } else _delay_ms(1000);
        } else if (direction==4){
                //object on R, turn L, inc R step, dec L Step
                PORTD.OUT=0x10;
                PORTD.OUT=0x08;
                if (move)
                {
                if(!objectFound){
                        turnLeftHard();
                } else turnLeft();
                turnLeft();
                } else _delay_ms(1000);

        } else if (direction==6){
                //object on L, turn R, inc L step, dec R Step

                PORTD.OUT=0x40;
                PORTD.OUT=0x80;
                if (move){
```

```
				if(!objectFound){
					turnRightHard();
				} else turnRight();
				turnRight();
				} else _delay_ms(1000);
			}else if(direction==9){
			//Back up

			if (lastLR==4){
				PORTD.OUT=0x18;
				if (move){
				turnLeftHard();
				turnLeftHard();
				if (direction1==9){
				turnLeftHard();
				}
				}else _delay_ms(1000);
				//nextState();
			} else {
				PORTD.OUT=0xC0;
				if (move){
				turnRightHard();
				turnRightHard();
				if (direction1==9){
				turnRightHard();
				}
				}else _delay_ms(1000);
			}
				//_delay_ms(1000);

		}else if(direction==1){
			//not sure what to do
			cylon();


		}


		}
		return 0;
}


void talkToLaptop(void){
		//talk to laptop vision system
		//PORTB.OUT=0x0f;
		uint8_t msgCount=0;
		directionFromCamera=0;
		objectFound=0;
		char xmsg="";
		//char xmsg[10];


		if (dataInBufE1()){
			msgCount++;
			fscanf(&Xbee_str,"%c",&xmsg);
			//fprintf(&USB_str," Xbee Message! %c \r \n", xmsg);
			//_delay_ms(500);
```

```
		countSinceLastMsg=0;
	}

	if (dataInBufE1()){
		msgCount++;
		fscanf(&Xbee_str,"%c",&xmsg);
		//fprintf(&USB_str," Xbee Message! %c \r \n", xmsg);
		//_delay_ms(500);
	}
	if (dataInBufE1()){
		msgCount++;
		fscanf(&Xbee_str,"%c",&xmsg);
		//fprintf(&USB_str," Xbee Message! %c \r \n", xmsg);
		//_delay_ms(500);
	}

	if(msgCount==1){
		directionFromCamera=5; objectFound=1;
		fprintf(&USB_str," Direction from camera from LT %i \r \n",
directionFromCamera);
	} else if(msgCount==2){
		directionFromCamera=4; objectFound=1;
		fprintf(&USB_str," Direction from camera from LT%i \r \n",
directionFromCamera);
	} else if(msgCount==3){
		directionFromCamera=6; objectFound=1;
		fprintf(&USB_str," Direction from camera from LT %i \r \n",
directionFromCamera);
	}

}


void directionMessage(void){

}


void turnLeft(void){
	//object on R, turn L, inc R step, dec L Step
	uint8_t dTl=3;
	uint8_t dTr=12;

		//Front Right leg
		setServoAngle(2,sZ+dT*4);
		setServoAngle(6,sZ+dK*2);
		//_delay_ms(200);

		//shift body forward
		setServoAngle(1,sZ-dT);
		setServoAngle(3,sZ+dT);

		//back Left Leg  at end usually
		setServoAngle(9,sZ-dT*4);
		setServoAngle(18,sZ-dK);
		_delay_ms(100);

		//finish Front right leg movement
```

```
            setServoAngle(2,sZ-dTr);
            _delay_ms(200);
            setServoAngle(6,sZ);
            //_delay_ms(200);

            //back Left Leg
            setServoAngle(9,sZ+dTl*2);
            _delay_ms(100);
            setServoAngle(18,sZ);

       //Front Left leg
       setServoAngle(1,sZ-dT*4);
       setServoAngle(5,sZ-dK*2);
       //_delay_ms(100);

       //shift body forward
       setServoAngle(2,sZ+dT);//90
       setServoAngle(9,sZ-dT);//0

       //back Right Leg
       setServoAngle(3,sZ+dT*4);
       setServoAngle(7,sZ+dK);
       _delay_ms(100);

       //finish Front Left movement
       setServoAngle(1,sZ+dTl);
       _delay_ms(100);
       setServoAngle(5,sZ);
       //_delay_ms(200);
       //setServoAngle(1,sZ+dTl);
       //_delay_ms(20);

       //back Right Leg
       setServoAngle(3,sZ-dTr*2);
       _delay_ms(200);
       setServoAngle(7,sZ);
       //_delay_ms(200);
       nextState();


}

void turnRight(void){
       //object on L, turn R, inc L step, dec R Step
       uint8_t dTl=12;
       uint8_t dTr=3;
       //Front Left leg
       setServoAngle(1,sZ-dT*4);
       setServoAngle(5,sZ-dK*2);
       //_delay_ms(100);

       //shift body forward
       setServoAngle(2,sZ+dT);//90
       setServoAngle(9,sZ-dT);//0

       //back Right Leg
       setServoAngle(3,sZ+dT*4);
       setServoAngle(7,sZ+dK);
```

```
	//_delay_ms(100);

	//finish Front Left movement
	setServoAngle(1,sZ+dTl);
	_delay_ms(200);
	setServoAngle(5,sZ);
	_delay_ms(200);

	//back Right Leg
	setServoAngle(3,sZ-dTr*2);
	_delay_ms(100);
	setServoAngle(7,sZ);
	//_delay_ms(200);
	nextState();

	//Front Right leg
	setServoAngle(2,sZ+dT*4);
	setServoAngle(6,sZ+dK*2);
	//_delay_ms(200);


	//shift body forward
	setServoAngle(1,sZ-dT);
	setServoAngle(3,sZ+dT);

	//back Left Leg
	setServoAngle(9,sZ-dT*4);
	setServoAngle(18,sZ-dK);
	//_delay_ms(100);

	//finish Front right leg movement
	setServoAngle(2,sZ-dTr);
	_delay_ms(100);
	setServoAngle(6,sZ);
	//_delay_ms(200);

	//back Left Leg
	setServoAngle(9,sZ+dTl*2);
	_delay_ms(100);
	setServoAngle(18,sZ);
	//_delay_ms(300);
}


void goForward(void){
				//Straight
				uint8_t m_Back=3;
				//Front Left leg
				setServoAngle(1,sZ-dT*4);  //4
				setServoAngle(5,sZ-dK*2);
				//setServoAngle(5,sZ+dK*2);
				//setServoAngle(1,sZ-dT*4);

				//_delay_ms(100);

				//shift body forward
				setServoAngle(2,sZ+dT);//90
				setServoAngle(9,sZ-dT);//0
```

```
                //back Right Leg
                setServoAngle(3,sZ+dT*m_Back);
                setServoAngle(7,sZ+dK);
                //_delay_ms(100);

                //finish Front Left movement
                setServoAngle(1,sZ+dT+5);  //dT
                _delay_ms(200);
                setServoAngle(5,sZ);
                _delay_ms(100);

                //back Right Leg
                setServoAngle(3,sZ-dT*2);
                _delay_ms(100);
                setServoAngle(7,sZ);
                //_delay_ms(200);

                nextState();

                //Front Right leg
                setServoAngle(2,sZ+dT*4);
                setServoAngle(6,sZ+dK*2);
                _delay_ms(100);

                //shift body forward
                setServoAngle(1,sZ-5); //-dT
                setServoAngle(3,sZ+5); //+dT

                //back Left Leg
                setServoAngle(9,sZ-dT*m_Back);
                setServoAngle(18,sZ-dK);
                //_delay_ms(100);

                //finish Front right leg movement
                setServoAngle(2,sZ-7);  //-dT
                _delay_ms(200);
                setServoAngle(6,sZ);
                _delay_ms(200);

                //back Left Leg
                setServoAngle(9,sZ+dT*3); //2
                _delay_ms(100);
                setServoAngle(18,sZ);
                //_delay_ms(300);


}


void turnLeftHard(void){
       //object on R, turn L, inc R step, dec L Step
       uint8_t dTl=5;
       uint8_t dTr=10;
       //Front Right leg
       setServoAngle(2,sZ+dT*4);
       setServoAngle(6,sZ+dK*2);
       _delay_ms(100);
```

```
	//shift body forward
	setServoAngle(1,sZ-dT);
	setServoAngle(3,sZ+dT);

	//finish Front right leg movement
	setServoAngle(2,sZ-dTr);
	_delay_ms(200);
	setServoAngle(6,sZ);
	_delay_ms(200);
	nextState();

	//Front Left leg
	setServoAngle(1,sZ-dT*3);  //3
	setServoAngle(5,sZ-dK*2);
	//_delay_ms(300);

	//back Left Leg
	setServoAngle(3,sZ+dT*4);
	setServoAngle(7,sZ+dK);
	_delay_ms(100);
	setServoAngle(3,sZ-dTr);
	_delay_ms(200);
	setServoAngle(7,sZ);
	_delay_ms(100);

	//finish Front Left movement
	setServoAngle(5,sZ);
	_delay_ms(20);
	setServoAngle(1,sZ+dTl);  //+dTl
	_delay_ms(20);

	//shift body forward
	setServoAngle(2,sZ+dT);//90
	setServoAngle(9,sZ-dT);//0
	//setServoAngle(9,sZ-dT);//0

}


void turnRightHard(void)
{
	//object on L, turn R, inc L step, dec R Step
	uint8_t dTl=10;
	uint8_t dTr=5;

	//Front Left leg
	setServoAngle(1,sZ-dT*4);
	setServoAngle(5,sZ-dK*2);
	//_delay_ms(300);

	//shift body forward
	setServoAngle(2,sZ+dT);//90
	setServoAngle(9,sZ-dT);//0


	//finish Front Left movement
	setServoAngle(1,sZ+dTl);
```

```
        _delay_ms(200);
        setServoAngle(5,sZ);
        _delay_ms(100);


        //Front Right leg
        setServoAngle(2,sZ+dT*3);   //4 ususally
        setServoAngle(6,sZ+dK*2);
        _delay_ms(100);

        //back Left Leg
        setServoAngle(9,sZ-dT*4);
        setServoAngle(18,sZ-dK);
        //_delay_ms(100);
        //back Left Leg
        setServoAngle(9,sZ+dTl);
        _delay_ms(200);
        setServoAngle(18,sZ);


        //finish Front right leg movement
        setServoAngle(2,sZ-0);  //-dTr
        _delay_ms(20);
        setServoAngle(6,sZ);
        //_delay_ms(200);
        nextState();


        //shift body forward
        setServoAngle(1,sZ-dT);
        setServoAngle(3,sZ+dT);

}

void nextState(void)
{
        // if large increase in new sensor values, something is probably in the way
        uint8_t large_increase=75;
        uint8_t sonar_Filtered_old=sonar_Filtered;
        uint8_t rightIR_Filtered_old=rightIR_Filtered;
        uint8_t leftIR_Filtered_old=leftIR_Filtered;

        //Move sensing to ISR at 10 Hz
        uint8_t sensorDirection;
        uint8_t sonar_Filter[]={0,0,0,0};
        uint8_t rightIR_Filter[]={0,0,0,0};
        uint8_t leftIR_Filter[]={0,0,0,0};
        uint8_t i, test;
        senseturn=0;

        for (i=0;i<4;i++){
        test =readSonar();
        sonar_Filter[3]=sonar_Filter[2];
        sonar_Filter[2]=sonar_Filter[1];
        sonar_Filter[1]=sonar_Filter[0];
        sonar_Filter[0]=test;
        //sonar_Filtered=((2*sonar_Filter[0]+sonar_Filter[1]+sonar_Filter[2])/6);
```

```
	test =readIR_R();
	rightIR_Filter[3]=rightIR_Filter[2];
	rightIR_Filter[2]=rightIR_Filter[1];
	rightIR_Filter[1]=rightIR_Filter[0];
	rightIR_Filter[0]=test;
	//rightIR_Filtered=(4/5)*rightIR_Filter[0]+(1/5)*rightIR_Filter[1];
	//rightIR_Filtered=((4*rightIR_Filter[0]+rightIR_Filter[1]+rightIR_Filter[2])/6);

	test =readIR_L();
	leftIR_Filter[3]=leftIR_Filter[2];
	leftIR_Filter[2]=leftIR_Filter[1];
	leftIR_Filter[1]=leftIR_Filter[0];
	leftIR_Filter[0]=test;
	//leftIR_Filtered=(4/5)*leftIR_Filter[0]+(1/5)*leftIR_Filter[1];
	//leftIR_Filtered=((4*leftIR_Filter[0]+leftIR_Filter[1]+leftIR_Filter[2])/6);
	}

	//sonar_Filtered=((3*sonar_Filter[0]+sonar_Filter[1]+sonar_Filter[2]+sonar_Filter[
3])/6);
	rightIR_Filtered=((5*rightIR_Filter[0]+rightIR_Filter[1]+rightIR_Filter[2]+rightIR
_Filter[3])/8);
	leftIR_Filtered=((5*leftIR_Filter[0]+leftIR_Filter[1]+leftIR_Filter[2]+leftIR_Filt
er[3])/8);

	//rightIR_Filtered=((3*rightIR_Filter[0]+rightIR_Filter[1])/4);
	//leftIR_Filtered=((3*leftIR_Filter[0]+leftIR_Filter[1])/4);

	sonar_Filtered=(sonar_Filter[0]+sonar_Filter[1]+sonar_Filter[2]+sonar_Filter[3])>>
2;
	//rightIR_Filtered=(rightIR_Filter[0]+rightIR_Filter[1]+rightIR_Filter[2]+rightIR_
Filter[3])>>2;
	//leftIR_Filtered=(leftIR_Filter[0]+leftIR_Filter[1]+leftIR_Filter[2]+leftIR_Filte
r[3])>>2;


	//Sensor direction logic
	if (( sonar_Filtered< sonarThr_L2)||(sonar_Filtered>sonarThr_U2)){//((
sonar_Filtered> sonarThr_L)&&(sonar_Filtered<sonarThr_U)){
		//IR sensors are higher or lower than upper or lower threshold turn
		if ((rightIR_Filtered < rightIRThr_L)||(leftIR_Filtered <
leftIRThr_L)||(rightIR_Filtered > rightIRThr_U)||(leftIR_Filtered >
leftIRThr_U)){//(rightIR_Filtered>1.2*leftIR_Filtered){

			if ((rightIR_Filtered>1.5*leftIR_Filtered)||(rightIR_Filtered <
rightIRThr_L)){
				//Left
				//PORTR.OUT=0x00;
				direction=4;
				lastLR=4;
			}
			else if ((leftIR_Filtered>1.5*rightIR_Filtered)||(leftIR_Filtered <
leftIRThr_L)){
				//PORTR.OUT=0x00;
				direction=6;
				lastLR=6;

				} else {
					//PORTR.OUT=0x00;
```

```
                                    direction=9;
                              }

                        if(((leftIR_Filtered < leftIRThr_L)&&(rightIR_Filtered <
rightIRThr_L))||((leftIR_Filtered > 2*leftIRThr_U)&&(rightIR_Filtered >
2*rightIRThr_U))){
                              direction=9;

                        }
                        /*
                        if(rightIR_Filtered < rightIRThr_L){
                              direction=4;
                              lastLR=4;
                        }
                              */

                  } else {

                        direction=5;
                  }

            }else {

                  //PORTR.OUT=0xFF;
                  direction=9;
            }
            //sonar gets last word
            if (( sonar_Filtered> sonarThr_L2)&&(sonar_Filtered<sonarThr_U2)){
                  direction=9;
            }
            sensorDirection=direction;

            //check for large increase
            if(fabs(leftIR_Filtered-leftIR_Filtered_old)>large_increase){
                  sensorDirection=9; lastLR=6; senseturn=1;
                  fprintf(&USB_str,"Turn Right! ");
            }
            if(fabs(rightIR_Filtered-rightIR_Filtered_old)>large_increase){
                  sensorDirection=9; lastLR=4; senseturn=1;
                  fprintf(&USB_str,"Turn Left!");
            }
            if(fabs(sonar_Filtered-sonar_Filtered_old)>large_increase){
                  sensorDirection=9; senseturn=1;
                  fprintf(&USB_str,"Turn Around! ");
            }

            //if((direction+direction1+direction2+direction3)>32) {
            //      senseturn=1;
            //}

            // Arbitrator logic
            if(objectFound){//&&(!senseturn)){

                  // if object found by camera, go that direction
                  PORTB.OUT=0x0f;
                  if (countSinceLastMsg<3){
                        direction=directionFromCamera;
                        fprintf(&USB_str,"Camera: %d \r\n",directionFromCamera);
```

```
			}else objectFound=0;
		//countSinceLastMsg=0;

		} else{

			//if object not found, go in last known dir or use sensor direction
if no msg in 4 cycles
			if (countSinceLastMsg<3){
				direction=direction1;
			} else if((countSinceLastMsg>2)&&(countSinceLastMsg<6)){

				direction=1;
			} else{
				direction=1;//sensorDirection;
				PORTB.OUT=0x00;
				if ((countSinceLastMsg>50)&&(countSinceLastMsg<250)) { //50
and 500
					PORTB.OUT=0x00;
					direction=sensorDirection;
				}
			}
		}
		//If there was a large increase in one of the sensor, turn
		if(senseturn&&(countSinceLastMsg>3)){
			fprintf(&USB_str," bc large sensor increase! \r\n");
			direction=sensorDirection;
		}

	countSinceLastMsg++;
	//save last sequence of direction
	direction3=direction2; direction2=direction1; direction1=direction;

	fprintf(&USB_str,"Count %d \r\n",countSinceLastMsg);
	fprintf(&USB_str,"sD %d Direction %d  Object %d Sonar %d Left %d Right %d LastLR:
%d\r\n",sensorDirection, direction, objectFound, sonar_Filtered, leftIR_Filtered,
rightIR_Filtered,lastLR);

	//direction=5;

	//lastLR=4;
}

void zeroServos(void){
//zero all servos, calibrate();
//move legs to 0 positionf
setServoAngle(1,sZ);
setServoAngle(2,sZ);
setServoAngle(9,sZ);
setServoAngle(3,sZ);
cylon();
setServoAngle(5,sZ);
setServoAngle(6,sZ);
setServoAngle(7,sZ);
setServoAngle(18,sZ);
cylon();

//shift body forward (thighs back) for first step
setServoAngle(1,sZ-dT*.5);
```

```
setServoAngle(2,sZ+dT*.5);
setServoAngle(3,sZ+dT*.5);
setServoAngle(9,sZ-dT*.5);
cylon();
}


void cylon(void){
	// cylon LED sequence with 1000 ms delay
	_delay_ms(100);
	PORTD.OUT=0x20;
	_delay_ms(100);
	PORTD.OUT=0x40;
	_delay_ms(100);
	PORTD.OUT=0x80;
	_delay_ms(100);
	PORTD.OUT=0x40;
	_delay_ms(100);
	PORTD.OUT=0x20;
	_delay_ms(100);
	PORTD.OUT=0x10;
	_delay_ms(100);
	PORTD.OUT=0x08;
	_delay_ms(100);
	PORTD.OUT=0x10;
	_delay_ms(100);
	PORTD.OUT=0x20;
	_delay_ms(100);
}

void cylon_slow(void){
	// cylon LED sequence with 1000 ms delay
	_delay_ms(300);
	PORTD.OUT=0x20;
	_delay_ms(300);
	PORTD.OUT=0x40;
	_delay_ms(300);
	PORTD.OUT=0x80;
	_delay_ms(300);
	PORTD.OUT=0x40;
	_delay_ms(300);
	PORTD.OUT=0x20;
	_delay_ms(300);
	PORTD.OUT=0x10;
	_delay_ms(300);
	PORTD.OUT=0x08;
	_delay_ms(300);
	PORTD.OUT=0x10;
	_delay_ms(300);
	PORTD.OUT=0x20;
	_delay_ms(300);
}

/*
ISR(TCF0_OVF_vect){
	//PORTR.DIRSET=0x02;
	//_delay_ms(1000);
	PORTR.OUTTGL = 0xff;
```

```
        fprintf(&USB_str,"%d \r\n", rightIR);//leftIR,rightIR);
}
*/
```

B. Color Tracking Code

Sample code adapted from Josh Weaver

```
#include <highgui.h>
#include <cv.h>
#include "stdafx.h"
#include <dos.h>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <Windows.h>
#include <string>
#include <sstream>
#include "SerialPort.h"


// Maths methods
//#define max(a, b) ((a) > (b) ? (a) : (b))
//#define min(a, b) ((a) < (b) ? (a) : (b))
#define abs(x) ((x) > 0 ? (x) : -(x))
#define sign(x) ((x) > 0 ? 1 : -1)

// Step mooving for object min & max
#define STEP_MIN 5
#define STEP_MAX 100

IplImage *image;
//CvFont font1;
//cvInitFont(&font1, CV_FONT_HERSHEY_SIMPLEX, 0.4, 0.4, 0, 1, 8);

// Position of the object we overlay
CvPoint objectPos = cvPoint(-1, -1);
// Color tracked and our tolerance towards it
int h = 0, s = 0, v = 0, tolerance = 4;
char tbName[50]="Filter Selection";
int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;
int go=0;


void tbCallBack(int, void*)
{
}

void createTrackBar(){
```

```
  // cvNamedWindow("Mask", CV_WINDOW_AUTOSIZE);
   // cvMoveWindow("Mask", 650, 0);
       cvNamedWindow(tbName,0);
       cvMoveWindow(tbName,650,350);
       char tbNameMem[50];
       sprintf(tbNameMem,"H_MIN",H_MIN);
       cv::createTrackbar("H_MIN",tbName,&H_MIN, H_MAX, tbCallBack);

}

/*
 * Transform the image into a two colored image, one color for the
color we want to track, another color for the others colors
 * From this image, we get two datas : the number of pixel detected,
and the center of gravity of these pixel
 */
CvPoint binarisation(IplImage* image, int *nbPixels) {
        int x, y;
//        CvScalar pixel;
        IplImage *hsv, *mask;
        IplConvKernel *kernel;
        int sommeX = 0, sommeY = 0;
        *nbPixels = 0;

        // Create the mask &initialize it to white (no color detected)
        mask = cvCreateImage(cvGetSize(image), image->depth, 1);

        // Create the hsv image
        hsv = cvCloneImage(image);
        cvCvtColor(image, hsv, CV_BGR2HSV);
        // We create the mask
        cvInRangeS(hsv, cvScalar(h - tolerance -1, s - tolerance, 0), cvScalar(h +
tolerance -1, s + tolerance, 255), mask);
        // Create kernels for the morphological operation
        kernel = cvCreateStructuringElementEx(5, 5, 2, 2, CV_SHAPE_ELLIPSE);

        // Morphological opening (inverse because we have white pixels on black
background)
        cvDilate(mask, mask, kernel, 4);
        cvErode(mask, mask, kernel, 4);


        // We go through the mask to look for the tracked object and get its gravity
center
        for(x = 0; x < mask->width; x++) {
                for(y = 0; y < mask->height; y++) {

                        // If its a tracked pixel, count it to the center of gravity's
calcul
                        if(((uchar *)(mask->imageData + y*mask->widthStep))[x] == 255) {
                                sommeX += x;
                                sommeY += y;
                                (*nbPixels)++;
                        }
                }
        }

        // Show the result of the mask image
```

```
        cvShowImage("Mask", mask);

        // We release the memory of kernels
        cvReleaseStructuringElement(&kernel);

        // We release the memory of the mask
        cvReleaseImage(&mask);
        // We release the memory of the hsv image
            cvReleaseImage(&hsv);

        if(*nbPixels > 0)
                return cvPoint((int)(sommeX / (*nbPixels)), (int)(sommeY / (*nbPixels)));
        else
                return cvPoint(-1, -1);

              if (!go){
              printf("GO!\n");
              }
              go=1;
}

void addObjectToVideo(IplImage* image, CvPoint objectNextPos, int nbPixels) {

        int objectNextStepX, objectNextStepY;

        // Calculate circle next position (if there is enough pixels)
        if (nbPixels > 10) {

                // Reset position if no pixel were found
                if (objectPos.x == -1 || objectPos.y == -1) {
                        objectPos.x = objectNextPos.x;
                        objectPos.y = objectNextPos.y;
                }

                // Move step by step the object position to the desired position
                if (abs(objectPos.x - objectNextPos.x) > STEP_MIN) {
                        objectNextStepX = max(STEP_MIN, min(STEP_MAX, abs(objectPos.x - 
objectNextPos.x) / 2));
                        objectPos.x += (-1) * sign(objectPos.x - objectNextPos.x) * 
objectNextStepX;
                }
                if (abs(objectPos.y - objectNextPos.y) > STEP_MIN) {
                        objectNextStepY = max(STEP_MIN, min(STEP_MAX, abs(objectPos.y - 
objectNextPos.y) / 2));
                        objectPos.y += (-1) * sign(objectPos.y - objectNextPos.y) * 
objectNextStepY;
                }

        // -1 = object isn't within the camera range
        } else {

                objectPos.x = -1;
                objectPos.y = -1;

        }

        // Draw an object (circle) centered on the calculated center of gravity
        if (nbPixels > 10)
```

```
              //cvDrawCircle(image, objectPos, 15, CV_RGB(255, 0, 0), -1);
            cvCircle(image, objectPos, 20, CV_RGB(0,0,255),2);
            cvLine(image, cvPoint(objectPos.x,objectPos.y+20),
            cvPoint(objectPos.x,objectPos.y-20),CV_RGB(0,0,255),2);
            cvLine(image, cvPoint(objectPos.x+20,objectPos.y), cvPoint(objectPos.x-
20,objectPos.y),CV_RGB(0,0,255),2);
                          //cvPutText(image,"Target:", objectPos,0,CV_RGB(0,0,255));
        // We show the image on the window
        cvShowImage("Color Tracking", image);

}

void getObjectColor(int event, int x, int y, int flags, void *param = NULL) {

        // Vars
        CvScalar pixel;
        IplImage *hsv;

        if(event == CV_EVENT_LBUTTONUP)        {

                // Get the hsv image
                hsv = cvCloneImage(image);
                cvCvtColor(image, hsv, CV_BGR2HSV);

                // Get the selected pixel
                pixel = cvGet2D(hsv, y, x);

                // Change the value of the tracked color with the color of the selected
pixel
                h = (int)pixel.val[0];
                s = (int)pixel.val[1];
                v = (int)pixel.val[2];

                            printf("HSV: %d %d %d \n",h,s,v);
                // Release the memory of the hsv image
                    cvReleaseImage(&hsv);

        }

}


int main() {

              // initialize serial port class and open port
              Serial* SP = new Serial();
              while (!(SP->IsConnected()))
              {
                     printf("Trying to connect... \r\n");
                     Sleep(500);
                     delete SP;
                     Serial* SP = new Serial();
                     //connected=SP->IsConnected;
              }


              if (SP->IsConnected()){
```

```
                printf("\r \n Connected! \r \n\n\n");

        //Serial::Serial();
        char incomingData[256]="";  //in data buffer
        char outData[256]="";  //out data buffer
        int dataLength = 256;
        int readResult = 0;


  CvCapture *capture;
  // Key for keyboard event
  char key = 'd';
        int direction;

  // Number of tracked pixels
  int nbPixels, i=1;
  // Next position of the object we overlay
  CvPoint objectNextPos;

  // Initialize the video Capture (200 => CV_CAP_V4L2)
 // Droid Cam from ManyCam=0; Laptop Webcam =1;
   capture = cvCaptureFromCAM(0);
  //capture = cvCreateFileCapture("http://192.168.1.106:8080/shot.jpg");

  // Check if the capture is ok
      if (!capture) {
          printf("Can't initialize the video capture.\n");
          return -1;
   }


  // Create the windows
        createTrackBar();
  cvNamedWindow("Color Tracking", CV_WINDOW_AUTOSIZE);
  cvNamedWindow("Mask", CV_WINDOW_AUTOSIZE);
  cvMoveWindow("Mask", 650, 0);
        cvMoveWindow("Color Tracking", 650, 350);

  // Mouse event to select the tracked color on the original image
  cvSetMouseCallback("Color Tracking", getObjectColor);

  // While we don't want to quit
  while(key != 'Q' && key != 'q') {
                        //if (go){
          // We get the current image
          image = cvQueryFrame(capture);

          // If there is no image, we exit the loop
          if(!image) continue;

          objectNextPos = binarisation(image, &nbPixels);
          addObjectToVideo(image, objectNextPos, nbPixels);

          // We wait 10 ms
          key = cvWaitKey(10);

                        //readResult = SP->ReadData(incomingData,dataLength);
                        //Decide if object is found and send position
```

```
                            //Note: send Y position because Droid is mounted sideways!

                            //turn camera
                            if ((i>15)){
                                   printf("X: %d %d\n",objectNextPos.x, nbPixels);
                                   readResult = SP->ReadData(incomingData,dataLength);
                                   printf("Bytes read: (-1 means no data available)
%i\n",readResult);
                                   //printf("Incoming Data: %c\n",incomingData);
                                   if(nbPixels>900){
                                          if(nbPixels>130000){
                                                 //printf("Too close to target, stop
\r\n");
                                          } else{
                                          //send X direction to laptop
                                          if(objectNextPos.x>410){
                                                 direction=4;
                                                 outData[0]='ll';
                                                 SP->WriteData(outData, 2);
                                          }else if (objectNextPos.x<250) {
                                                 direction=6;
                                                 outData[0]='rrr';
                                                 SP->WriteData(outData, 3);
                                          }else
if((objectNextPos.x>250)&&(objectNextPos.x<450)){
                                                 direction=5;
                                                 outData[0]='f';
                                                 //cvWaitKey(600);
                                                 SP->WriteData(outData, 1);

                                          } else{

                                                 printf("Target not found! \r\n");
                                          }
                                          printf("Send: %i %s \r\n", direction, outData);
                                          }
                                   }
                                   i=1;
                            }else i++;
                            //}

        }


              //Release serial port
              if(SP->IsConnected()){
                     SP->~Serial();
                     printf("Serial Port closed.");
              }


        // Destroy the windows we have created
        cvDestroyWindow("Color Tracking");
        cvDestroyWindow("Mask");

        // Destroy the capture
        cvReleaseCapture(&capture);
```

```
            }else printf("Could not open serial port!");

            Sleep(1000);
        return 0;

}


Serial::Serial()//char *portName)
{
    //We're not yet connected
    this->connected = false;

       //LPCWSTR p=*portName;
//     printf(TEXT("COM5"));
    //Try to connect to the given port throuh CreateFile
    this->hSerial = CreateFile(TEXT("\\\\.\\COM12"),//TEXT(portName),
            GENERIC_READ | GENERIC_WRITE,
            0,
            NULL,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

    //Check if the connection was successfull
    if(this->hSerial==INVALID_HANDLE_VALUE)
    {
        //If not success full display an Error
        if(GetLastError()==ERROR_FILE_NOT_FOUND){

            //Print Error if neccessary
            printf("ERROR: Handle was not attached. Reason: COM not available.\n");//,
portName);

        }
        else
        {
            printf("ERROR!!!");
        }
    }
    else
    {
        //If connected we try to set the comm parameters
        DCB dcbSerialParams = {0};

        //Try to get the current
        if (!GetCommState(this->hSerial, &dcbSerialParams))
        {
            //If impossible, show an error
            printf("failed to get current serial parameters!");
        }
        else
        {
            //Define serial connection parameters for the arduino board
            dcbSerialParams.BaudRate=CBR_9600;
            dcbSerialParams.ByteSize=8;
            dcbSerialParams.StopBits=ONESTOPBIT;
            dcbSerialParams.Parity=NOPARITY;
```

```
            //Set the parameters and check for their proper application
            if(!SetCommState(hSerial, &dcbSerialParams))
            {
               printf("ALERT: Could not set Serial Port parameters");
            }
            else
            {
                //If everything went fine we're connected
                this->connected = true;
                //Sleep(200);
                            printf(" Serial Port Connected \r\n");
            }
        }
    }

}

Serial::~Serial()
{
    //Check if we are connected before trying to disconnect
    if(this->connected)
    {
        //We're no longer connected
        this->connected = false;
        //Close the serial handler
        CloseHandle(this->hSerial);
    }
}

int Serial::ReadData(char *buffer, unsigned int nbChar)
{
    //Number of bytes we'll have read
    DWORD bytesRead;
    //Number of bytes we'll really ask to read
    unsigned int toRead;

    //Use the ClearCommError function to get status info on the Serial port
    ClearCommError(this->hSerial, &this->errors, &this->status);

    //Check if there is something to read
    if(this->status.cbInQue>0)
    {
        //If there is we check if there is enough data to read the required number
        //of characters, if not we'll read only the available characters to prevent
        //locking of the application.
        if(this->status.cbInQue>nbChar)
        {
            toRead = nbChar;
        }
        else
        {
            toRead = this->status.cbInQue;
        }

        //Try to read the require number of chars, and return the number of read bytes on
success
        if(ReadFile(this->hSerial, buffer, toRead, &bytesRead, NULL) && bytesRead != 0)
```

```
        {
            return bytesRead;
        }

    }

    //If nothing has been read, or that an error was detected return -1
    return -1;

}


bool Serial::WriteData(char *buffer, unsigned int nbChar)
{
    DWORD bytesSend;

    //Try to write the buffer on the Serial port
    if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend, 0))
    {
        //In case it don't work get comm error and return false
        ClearCommError(this->hSerial, &this->errors, &this->status);

        return false;
    }
    else
        return true;
}

bool Serial::IsConnected()
{
    //Simply return the connection status
    return this->connected;
}


}
```

C. Sonar/IR Characterization

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

#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
//#include "adc.h"
//#include "motor.h"
#include "ADC_manual.h"

#define analogChannel1 0  //analog channel on port A ADC Channel 0
#define analogChannel2 1  //analog channel on port A ADC Channel 1
#define leftIRPin 0
```

```
#define rightIRPin 2
#define sonarPin 3
#define leftIRThreshold 1000
#define rightIRThreshold 1000
#define sonarThreshold 3000

//To be used for path planning and stability measure?
uint8_t leftIR;                  //analog distance value
uint8_t rightIR;
uint8_t sonar;
int8_t direction;
int8_t directionFromCamera;


int main(void)
{
	uint8_t test;

	direction=9;
	uint8_t sonar_Filtered=0;
	uint8_t rightIR_Filtered=0;
	uint8_t leftIR_Filtered=0;
	uint8_t sonar_Filter[]={0,0,0};
	uint8_t rightIR_Filter[]={0,0,0};
	uint8_t leftIR_Filter[]={0,0,0};

	clockInit();
	//adcInit(&ADCA);
	ATtinyServoInit();
	usartInit(&USARTC0,115200);//USB port
	sei();

	//Blink Debug LED
	PORTR.DIRSET=0x02;
	_delay_ms(1000);
	PORTR.OUT = 0xff;  //Inverted logic on port r!
	_delay_ms(1000);
	_delay_ms(1000);

	//ADC likes being here
	ADC_init();

	while(1){
		sonar =readSonar();
		sonar_Filter[2]=sonar_Filter[1];
		sonar_Filter[1]=sonar_Filter[0];
		sonar_Filter[0]=sonar;
		sonar_Filtered=((2*sonar_Filter[0]+sonar_Filter[1]+sonar_Filter[2])/6);

		rightIR =readIR_R();
		rightIR_Filter[2]=rightIR_Filter[1];
		rightIR_Filter[1]=rightIR_Filter[0];
		rightIR_Filter[0]=rightIR;
		//rightIR_Filtered=(4/5)*rightIR_Filter[0]+(1/5)*rightIR_Filter[1];

	rightIR_Filtered=((4*rightIR_Filter[0]+rightIR_Filter[1]+rightIR_Filter[2])/6);

		leftIR =readIR_L();
```

```
		leftIR_Filter[2]=leftIR_Filter[1];
		leftIR_Filter[1]=leftIR_Filter[0];
		leftIR_Filter[0]=leftIR;
		//leftIR_Filtered=(4/5)*leftIR_Filter[0]+(1/5)*leftIR_Filter[1];
		leftIR_Filtered=((4*leftIR_Filter[0]+leftIR_Filter[1]+leftIR_Filter[2])/6);




		if (sonar_Filtered<45){
		if (( sonar_Filtered> 30)&&(1)){
			PORTR.OUT=0x00;
			direction=0;
			//for (i=0;i<5;i++){
				//sonar[0]=0;
			//}
		}
		else {
			PORTR.OUT=0xFF;
			direction=9;
		}

		}else {

		if (rightIR_Filtered>leftIR_Filtered){

				if (rightIR_Filtered > 80){
					PORTR.OUT=0x00;
					direction=1;
				}

				else PORTR.OUT=0xFF;

		} else if (leftIR_Filtered>rightIR_Filtered){
		if ((leftIR_Filtered > 80)&&(!(leftIR_Filtered<20))){
			PORTR.OUT=0x00;
			direction=-1;
		}
		else {
			PORTR.OUT=0xFF;
		}
		}
		}


		fprintf(&USB_str,"%d \r\n", rightIR);//leftIR,rightIR);
		_delay_ms(500);
		//fprintf(&USB_str,"Direction %d Sonar %d Left %d Right %d\r\n", direction,
sonar_Filtered, leftIR_Filtered, rightIR_Filtered);


	}



}
```