Contents

Abstract

The goal of this project is to make a robot that can follow alongside a BMX rider, keep a camera pointed at him, and record the riding with the help of a laptop computer. For simplicity, this robot will be designed only to film flatland tricks, meaning it will never have to climb over obstacles. To accomplish this effectively, the robot will need to be able to move up to 10 mph. It will also need to keep a safe distance of 3 meters from the rider to avoid damage in case the rider falls. The robot is expected to run into a lot of things, so it will be designed to survive a 10mph collision. The camera will be mounted on 2 servos so it can rotate about 2 axes. In collision avoidance, the robot will avert its path towards the rider so it can continue filming. If the robot can't avoid a collision without getting too close to the rider, it will stop moving but continue filming.

Executive Summary

The robot is designed to follow along the side of a BMX rider with a camera pointed at him, to record the tricks. Since BMX riders ride up to 10 mph when doing tricks, the robot needs to be able to match that speed. This means relatively powerful motors are needed, along with LiPo batteries to meet the high current needs of powerful motors. LiPo batteries can be dangerous if used improperly so the appropriate precautions will be taken. The robot will use a 3-wheel mobile platform with a caster wheel in the back and power going to 2 wheels in the front. It will be steered by differential steering because it's cheap and simple to implement.

The robot has to be able to work outside. This means that it can't have sensors whose performance is affected by light intensity. One SONAR sensor will be used in front for collision avoidance. This will be the longest-range SONAR sensor that is feasible to put on the robot. This is to allow the robot to see obstacles sooner so it can do the calculations to avoid them in time. Another SONAR sensor will be mounted with the camera to check the distance to the rider. The camera itself is a webcam that works wirelessly with a nearby laptop computer. It will be mounted on a servo to allow it to pivot.

It is almost certain that this robot will crash into things during testing. Since I'd like to use the robot more than once, it will have to be sturdy enough to take a 10-mph collision. Birch wood might split when hit that hard. As a precaution, all wooden parts will be aligned so that the grain runs side to side, not front to back. There is a plastic bumper in the front of the robot to provide further protection. If birch wood proves to be simply too weak to take the collisions, the broken parts will be made out of sheet metal instead.

Introduction

A common problem among recreational BMXers is that what they think they look like and what they really look like don't match. If they remain ignorant of what they really look like, it often results in bad habits that must be corrected later on. It is much easier to correct these mistakes immediately than 6 months later when they find out that they're doing a trick wrong. Correcting mistakes immediately makes riding safer and more fun for the rider. When a rider can see exactly where he went wrong while doing a trick, he will be better suited to find out how to do it right, so he can land the trick the next time.

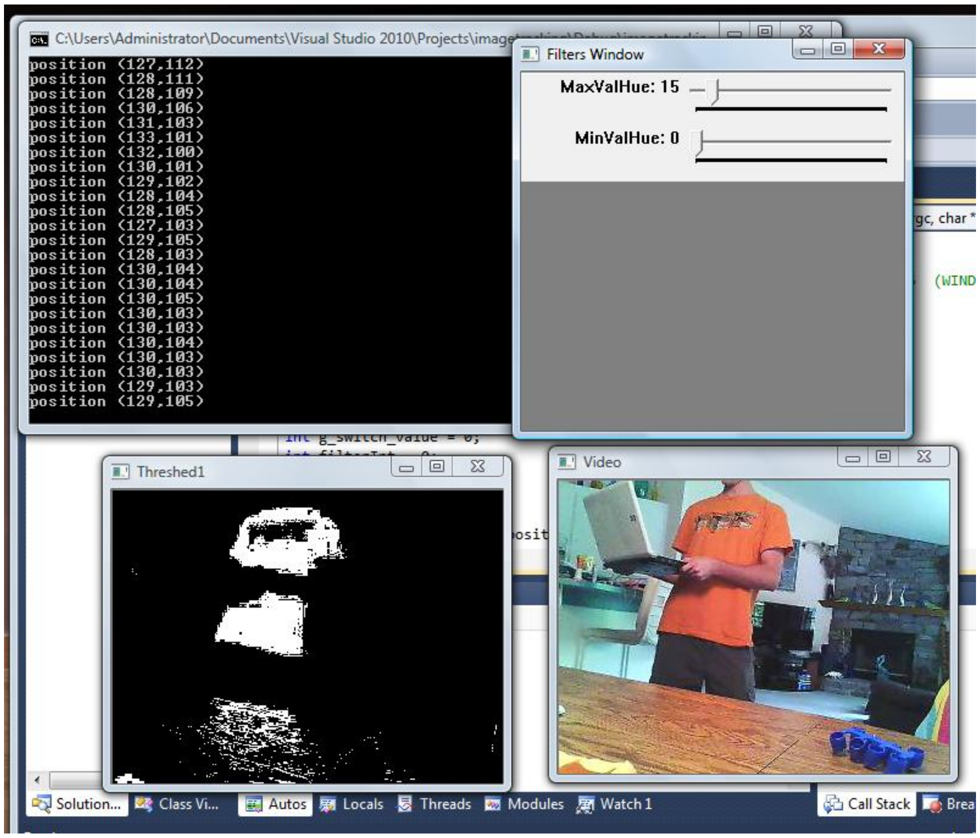
The purpose of this project is to make a robot that allows a rider to watch his tricks conveniently on a laptop, immediately after doing them. This gives him immediate feedback where he can critique his form. Ideally, the robot will modulate its speed to stay directly to the side of the rider, giving a perfect viewing angle at all times. The current method of filming BMX tricks involves a friend riding his bike next to you, with one hand on the handlebars and the other pointing a camera in your general direction. This doesn't allow the cameraman to adjust his speed, or aim the camera effectively. The robot will be able to do both so, if it works cleanly, it will be superior to the friend-on-a-bike method.

This report goes over the inner workings of the robot, how and why each part works the way it does, and the reasoning used behind each design decision. Each system and its function is described in detail.

Integrated System

This robot uses Sonar for obstacle avoidance and keeping its distance from its target, and a camera to track its target. One sonar is pointed forward and two are aimed at the target. The sonar are chained and operate sequentially to avoid interference. The camera broadcasts to the internet while a computer program analyzes the video and finds the current position of the rider. The program does this by detecting color within a certain hue range, and finding the average position of everything it detects. The hue range is set to match the color of the rider's shirt. Orange seems to work best. When he gets too far to either side of the screen, the computer sends a signal to the robot to either speed up or slow down to get him closer to the center, effectively matching the rider's speed. If the camera is detecting no rider, the computer sends a signal to the robot to stop its motors because it lost its target. The two sonar used for keeping the robot's distance operate based on the assumption that the rider is the closest thing to the robot. When they both get ranges below the lower bound, the robot turns left to increase its distance from the rider. When they're getting ranges above the upper bound, the robot turns right to get closer to the rider. Also, the difference between the two sonar readings is taken and used for calculating the robot's angle to the rider. When the robot is within the correct range to the rider, it works to minimize this difference by being square to the rider. All steering is done using differential steering, giving less power to the motor on whichever side it wants to turn to.

Because of the high speed, obstacle avoidance takes priority over everything. If the front sonar detects anything within 2.5 meters, it begins obstacle avoidance. If the rider is far enough away, the robot takes a hard right turn to try to dodge the obstacle. If the rider is too close (<1.5 m) the robot can't dodge the obstacle without getting dangerously close to the rider so it stops the motors to minimize collision speed. It is the rider's responsibility to make sure the robot has room to turn when riding toward an obstacle.



```
position (127,112)
position (128,111)
position (128,109)
position (130,106)
position (131,103)
position (133,101)
position (132,100)
position (130,101)
position (129,102)
position (128,104)
position (128,105)
position (127,103)
position (129,105)
position (128,103)
position (130,104)
position (130,104)
position (130,105)
position (130,103)
position (130,103)
position (130,104)
position (130,103)
position (130,103)
position (129,103)
position (129,105)
```

MaxValHue: 15

MinValHue: 0

Image tracking program detecting orange. Top left: average position of everything detected by the program, top right: Hue detection range, bottom left: everything the robot sees within its detection range shows up white, bottom right: Camera input.

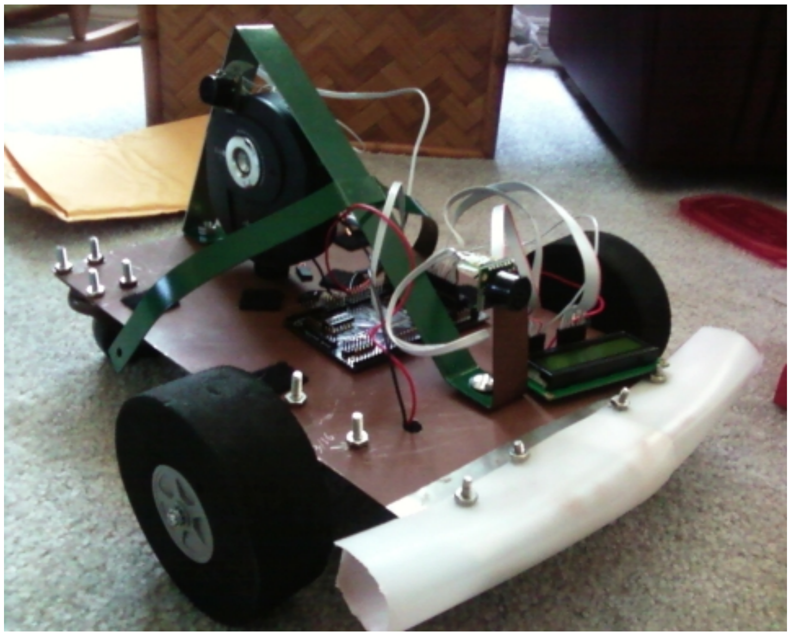
Mobile Platform

Most platforms in IMDL are made from birch wood because it's cheap, available, and can be machined into just about any shape you want thanks to the T-Tech machine. I considered using birch wood for my platform, but since the robot has to survive high-speed collisions, I chose to make it out of 20-gauge sheet steel instead. To protect from rusting, I spray-painted the platform. As a precaution for collisions, I installed a plastic bumper on the front of the robot, made from a cleverly repurposed shampoo bottle. In the middle of the platform is a 1-inch diameter hole, used for wires to go through to the bottom, and to make it easier to carry with one hand.

The back two wheels are casters, and the front two have the power going to them. Delicate electronic equipment is mounted to the platform by Velcro because it's easy to remove from the platform, and in the event of a crash, it can move around a bit. The exception to this is Sonar, which is mounted on angle brackets by bolts. This is because I don't want the Sonar sensors moving around.

A rollbar has been added to the platform to protect the most expensive parts of the robot in the event of a rollover. The rollbar protects the LiPo battery, camera, board, ESCs and one sonar sensor, a total of about 400 dollars worth of electronics.

Mobile Platform Picture



Due to the motors' complete lack of control, I got many chances to test this mobile platform's rigidity. It crashed many times and the robot came out unscathed. However, it almost seemed like an accomplishment when the robot managed to move far enough to collide into something.

Actuation

Big Brother is moved by two Beetle B04 motors from robotmarketplace.com, using differential steering. Differential steering was selected because it's simple to implement and also the easiest to program. These motors are driven by two Traxxas XL-10 ESCs, bought the day before media day from the Hobbytown in Jacksonville because they were the closest place that was open on Sunday and had ESCs that I could use on my robot. I had no other choice. The ESCs turned out to work very well. The motors didn't. They were easy to mount because ½ inch pipe straps fit them perfectly. The advantages end there. These motors ran at up to 2000 RPM, and were nearly impossible to control. They work perfectly with the wheels off the ground, but when you put them on the ground, they can't even move in a straight line, one motor stops turning, and the robot ends up spinning in circles. These motors single-handedly ruined an otherwise perfectly-functioning robot. Had I used slower motors with more torque, the robot would have worked perfectly. It would have been slow and lame, but it would have worked.

Sensors

This robot is designed to work outside, a requirement that makes any type of sensor whose performance is sensitive to light completely useless. The only type of obstacle avoidance sensor that's feasible outside is Sonar. Since the robot has to move fast, I'm using the longest range sonar I could find: the MaxSonar EZ1 from Sparkfun.com. At 3.3 volts, it can detect obstacles from up to 3.5 meters away. 5 volts makes it perform significantly better, allowing it to detect obstacles from up to 7 meters. To make it run at 5 volts takes some extra programming work so I will only do that if I absolutely need to.

Big Brother's special sensor is the system of tracking cameras with range finding. The cameras are the most important sensors on this robot. With stereoscopic vision, it is possible that this robot can function using only cameras. I am using the Cisco Linksys IP home monitoring camera. For depth perception, the robot will have a Sonar sensor mounted along with the camera, and, if there is time, a second camera used for stereoscopic vision. This will allow redundancy in the system so if one camera or the sonar sensor stops working, the robot can still function.

To communicate with the computer, the processor board is attached to a device that detects electromagnetic waves emitted by the computer. This is needed so the board will know where I am in relation to the robot, and when it needs to speed up and slow down.

If I have time, I will add a fourth sensor, either a bump sensor to detect collisions or a photoresistor that turns a flashlight on in the dark, allowing the robot to work at night.

Behaviors

With the wheels propped up off the ground, the robot does everything it was designed to do. It speeds up when I get ahead, slows down when I fall behind, and does the necessary turning to keep the robot square to the rider. That means nothing because the wheels are off the ground, not doing anything. With the wheels on the ground, it follows the rider for about 10 feet if you're lucky before spinning out. I would say something about the camera watching the spin out, but when the camera is moved around too much, it freezes up. The robot's spinning out is enough to freeze the camera.

Experimental layout/results

The robot works better than my presentation on media day made it look. The image tracking program works great, and every system on the robot works except the motors. The sonar give accurate readings and the robot reacts to them accordingly. This can be observed by watching the robot work perfectly when its wheels are off the ground. The fact that it works perfectly with its wheels off the ground turned out to be its downfall. Fast-moving robots are hard to test since you need an open area so I resorted to propping the wheels up off the ground for testing. This gave the illusion that my robot worked. It was only on media day morning that I found out my motors can't control themselves on the ground. It was too late to replace the motors so I was faced with the task of getting a robot to control itself at high speeds with crappy motors, in 5 hours. Nothing I did could make it work and it continued to spin itself out. In fact, the only thing wrong with this robot was the motors. If I were using slow-moving motors, it would have worked. If I were using higher quality motors that were still fast, it still might have worked. Instead, I was using cheap, fast-moving motors, which are poorly designed and made my robot look like a complete failure.

Conclusion

As stated before, besides the motors, the robot worked perfectly. The color detection program along with the camera worked much better than expected. Xbee communicators were hard to implement, but worked fine. The sonar worked nearly perfectly when chained together to avoid interference. Really, the only parts that don't work are the motors, and if the motors don't work, the entire robot looks terrible.

There are some limitations to use of this robot. It must be used on a smooth surface, which isn't much of a limitation since that's the only place anyone does flatland BMX. The bigger limitations are the fact that the camera needs a wireless network to work, and that you need a computer to process images. The Xbee communicators have a 1-mile range so you can't go further than that from the computer. Wireless routers tend to have a shorter range, and it's not like anyone is going to bring a wireless router and their computer with them when they go to ride BMX.

One thing I learned here was that if you're thinking about making a fast-moving robot for IMDL, don't. If you're thinking about buying any motor from robotmarketplace.com that starts with the word "beetle" don't. Slow robots are lame, but at least they work. When you try to make your robot fast, it becomes very difficult to control, and cheap motors make it even worse. It's also very frustrating when you spend so much time and money getting everything to work, and it turns out none of it actually works because your motors suck.

The only thing I would change about this robot is replacing the motors with slower ones with more torque. I was satisfied with the performance of everything else.

Appendices

Appendix A: Robot code (C)

```
#include <avr/io.h>
#include "avr_compiler.h"
#include "usart_driver.h"
#include "PVR.h"
```

```
#include <math.h>

#include <stdio.h>

#define USART USARTF0

#define USART_BAUD 11500

/**/#define SERIAL_UBBRVAL(baud) ((((F_CPU / 16) + (baud / 2)) / (baud)) - 1)



#define SAMPLE_SIZE 32

USART_data_t USART_data;

void usart_initialize(void)

{

        //pin 3 output

        PORTF.DIRSET = PIN3_bm;

        //pin2 input

        PORTF.DIRCLR = PIN2_bm;

//      USART_InterruptDriver_Initialize(&USART_data, &USART,
USART_DREINTLVL(3));

        //usartc0, 8 data bits, no parity, 1 stop bit

        USART_Format_Set(&USART, USART_CHSIZE_8BIT_gc,
USART_PMODE_DISABLED_gc, false);

        //ENABLE INTERRUPT

//      USART_RxdInterruptLevel_Set(USART_data.usart, USART_RXCINTLVL(3));
```

```
	//set baud rate

/**/USART_Baudrate_Set(&USART, 17, 0);


	//ENABLE RX AND TX

	USART_Rx_Enable(&USART);

	USART_Tx_Enable(&USART);

	//Enabel PMIC Interrupt level low

	//PMIC.CTRL |= PCMIC_LOLVLEX_bm;


	//enable global interrupts

	//sei();

}

inline void usart_tx_byte(char DataByte)

{

	int txrxVal = 0;


	while(1)

	{

		txrxVal = USARTF0_STATUS;
```

```
                txrxVal &= 0x20;;

                if(txrxVal == 0x20)

                {

                        USARTF0_DATA = DataByte;

                        break;

                }

        }

}


inline void usart_tx_string(char *StringPtr)

{

        int i = 0;

        while(StringPtr[i] != 0) //while not null terminator

        {

                usart_tx_byte(StringPtr[i]);

                i++;

        }

}
```

```
inline char usart_rx_byte(void) //Changed from char to int, changed back

{

                    char txrxVal =  USARTF0_STATUS;

                    txrxVal &= 0x80;

                    char data;

                    if(txrxVal == 0x80)

                    {

                              data = USARTF0_DATA;

                    }

                    else

                    {

                              data = 0;

                    }

                    return data;

}

/**

ISR(USARTC0_RXC_vect)

{

          USART_RXComplete(&USART_data);

}
```

```
ISR(USARTC0_DRE_vect)

{

	USART_DataRegEmpty(&USART_data);

}
**/

void main(void)
{
	xmegaInit();						//setup XMega
	delayInit();						//setup delay functions
	ServoCInit();						//setup PORTC Servos
	ServoDInit();						//setup PORTD Servos
	ADCAInit();							//setup PORTA
analong readings
	lcdInit();							//setup LCD on
PORTK
	lcdString("Big Brother Is");	//display "Big Brother Is Watching" on top line (Line 0) of
LCD
	lcdGoto(1,0);						//move LCD cursor to the
second line (Line 1) of LCD
	lcdString("Watching");				//display "Board Demo" on second
line
	PORTQ_DIR |= 0x01;					//set Q0 (LED) as
output
	usart_initialize(); //setup usart c0



	PORTJ_DIR |=0b11111111;
	PORTJ_OUT |= 0b11111111;
	PORTH_DIR |=0b10000000;


	int i;
	int j;
	int k;
	int r=0;
```

```
        int dif;
        int avpower=50;
        int rightpower=0;
        int leftpower=0;
        char data;
        int dataI;
        delay_ms(1000);
        while(r<200)
        {




        //Check sonar
                PORTH_OUT |= 0b10000000;        //Strobe first sensor's RX
                delay_us(50);                   //for 50 microseconds
                //Check sonar readings
                i=ADCA0();              // Check front-front sonar
                PORTH_OUT &= 0b01111111;
                delay_ms(50);
                j=ADCA2();              // Check back-side sonar
                delay_ms(50);
                k=ADCA4();              // Check front-side sonar
                delay_ms(50);
                dif=j-k;  //Calcs angle between target and robot, dif>0 if robot is turned away.
dif<0 if turned towards

//Check Camera
        //Undo this comment to make the camera work*******
                data = usart_rx_byte();                 //Read data from xbee
                //usart_tx_byte(0x4a);
                //dataI=atoi(data);
                //i=ADCA0();                                    //read sensor from ADC0 port
                //i=50;
                //j=ADCA2();

                dataI=data-'0';
                //lcdData(0x01);
                //lcdInt(i);                      //print value from Xbee port L
                //delay_ms(100);
                //lcdGoto(1,0);
                //lcdInt(dataI);
                if (data=='n' )  avpower=avpower;  //target in detection zone, maintain speed
                if(data=='g' || dif>800) avpower=avpower+5;  //target is getting ahead, speed up
                if(data=='s' || dif<-800) avpower=avpower-5;  //target is falling behind, slow down
                if(data=='x')   //Camera is running but detecting nothing.  Sit still.
                {
                avpower=0;

                }
                //**********end of make the camera work section*******/
                if (avpower>60) avpower=60;  //Cap speed at 60% power
                if (avpower<0) avpower=0;    //Keep speed out of the negative range.


        //Make decisions based on readings


                if(i>500)                   //no obstacles detected, drive like crazy
                {
                        if (k>1000)               //rider too far away, turn right
                                {
                                rightpower=avpower-5;
                                leftpower=avpower;
                                }
                        else if (k<500)           //rider too close, turn left
                                {
                                rightpower=avpower;
                                leftpower=avpower-5;
                                }
                        else// rider in range, keep going straight, maintain straightness
                                {
                                        if(dif>200)      //Robot is turned toward target
                                                {
                                                rightpower=avpower;
                                                leftpower=avpower-2;
                                                }
                                        else if(dif<-200) //robot is turned away from rider.
                                                {
```

```
                                        rightpower=avpower-2;
                                        leftpower=avpower;
                                        }
                                   else                    //Rider is square to the robot
                                        {
                                        rightpower=avpower;
                                        leftpower=avpower;
                                        }

                              }
                    }



                    if(i<500)  //obstacle detected, avoid it
                    {
                         if (k<600)       //No room to turn, stop everything.
                              {
                              rightpower=0;
                              leftpower=0;
                              }
                         if (k>600)                 //Room to turn, hard right (yeah right, more like
          skid to a stop)
                              {
                              rightpower=avpower-5;
                              leftpower=avpower+5;
                              }
                    }
                    lcdData(0x01);
                    lcdInt(rightpower);
                    lcdGoto(1,0);
                    lcdInt(leftpower);




                    ServoD4(rightpower);
                    ServoD1(leftpower);

                    r++;                //Loop counter
```

```
                //print values


                lcdGoto(0,3);
                lcdInt(i);
                lcdGoto(0,8);
                lcdInt(j);
                lcdGoto(1,10);
                lcdInt(r);










        }
        //Done with robot.  Turn motors off.
        ServoD1(0);
        ServoD4(0);
}
```