Student Name: Matt Shockley
TAs: Mike Pridgen
Thomas Vermeer
Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

Final Report

OMNI

Table of Contents

Abstract

OMNI is an autonomous omni-directional computer vision robot designed to implement any number of computer vision applications. OMNI's movement made omni-directional through the use of 4 omni-directional wheels which allow for movement along two perpendicular axes. Using this drive system, OMNI is capable of moving in any direction without rotating, though it is also capable of rotating in place as well. OMNI is equipped with a single IP webcam which transmits image data to a remote machine, in this case a laptop computer, for image processing and computer vision algorithms. This is then translated into movement commands and sent via wireless serial link to OMNI's PVR board, where it is factored into an on-the-fly behavior algorithm. This configuration is ideal for implementing computationally expensive vision algorithms remotely, allowing the PVR board on OMNI to only handle the behavior algorithm. OMNI is also equipped with a ring of 8 infrared rangefinder sensors and 4 radial bump switches for obstacle detection and avoidance. Though not used in OMNI's final behaviors for this course, OMNI's camera is also mounted on a 360 degree servo and is able to pan on its own axis which will allow for some particularly interesting behaviors to be implemented in the future. I intended OMNI to be a platform with potential for growth and expansion and I believe that I have achieved my goal.

Executive Summary

My goal when creating OMNI was to create an extensible autonomous computer vision platform. OMNI is designed to tackle any variety of computer vision algorithms, but for this class he was programmed with a simple color tracking and following algorithm. OMNI derives its name from being an omni-directional robot using dual axis wheels which allow OMNI to roll across perpendicular axes.

OMNI is equipped with 8 infrared rangefinder sensors spaced along every edge of its octagonal chassis. It uses these for obstacle awareness and incorporates their readings into its behavior algorithms. OMNI is also equipped with 4 radial bump sensors which allow OMNI to detect if it has collided with an obstacle, and determine the ideal direction to move to get away from the object after a collision has occurred.

OMNI's special sensor is an IP webcam mounted in the center of its top platform. This webcam feeds a stream back to a remote machine where vision processing can occur. This machine then transmits commands to OMNI via a wireless serial link, thanks to a pair of linked xBee modules. This is OMNI's defining system, since vision processing is done remotely on a much more powerful machine (ideally), it is possible to perform complex vision algorithms while freeing up the PVR board mounted on OMNI to simply handle behaviors. OMNI's camera is also mounted on a 360 degree servo, which was not used for this class due to time constraints, but is definitely an area for future expansion.

Overall, OMNI is a successful project, held back only by having its vision algorithms run on a 3 year old laptop that was unable to handle this plus the overhead involved in the serial communication. As a proof of concept, OMNI is quite functional and operates extremely smoothly if linked to a fast machine, such as a multi-core desktop.

Introduction

OMNI represents the pinnacle of my studies in computer engineering, and has been a tremendous learning experience for me over the course of the semester. I have always been interested in computer vision and I wanted to build a robot which would have large potential for variability in applications in computer vision. Because of this, I decided to design a vision robot which used an IP webcam which would send image data to be processed on a more powerful remote machine, giving me the opportunity to run any variety of computer vision algorithm and preventing me from being limited by the specs of the PVR board or a proprietary camera board such as the CMU cam. Initially, I wanted to place this on a quadruped robot, but decided that I would probably spend more time on the mechanical engineering aspect of the robot, something not exactly in my studies. I still wanted to have an unorthodox method of locomotion, but I wanted something much simpler to control, and it was this line of thought that led me to omni-directional wheels. They were unusual enough to make me happy (I loved hearing "it looks like a crab!" when OMNI moved along a diagonal axis), and was also very simple to write servo control algorithms for. These omni-directional wheels then led me to design the robot chassis, 2 octagonal sandwiched platforms which allowed for an aesthetically pleasing symmetrical design. On the bottom platform I mounted the batteries and wheel servos, while on the top platform I mounted all of the electronics and sensors.

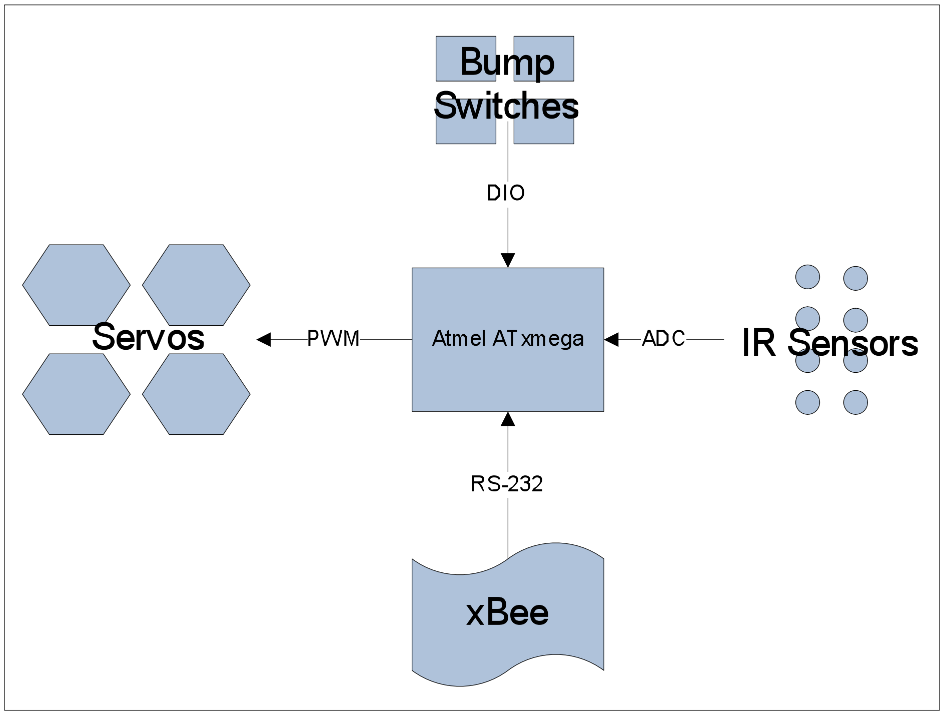
The behavior that I ultimately ended up implementing for this class was a simple color-following routine which used the webcam to track predetermined colors and then follow the object. This behavior runs in tandem with the obstacle avoidance routine, which takes precedence if there is a conflict. However, due to processor throughput limitations, even this algorithm ran rather slow on my external machine, a 3 year old laptop. This occasionally introduced some nasty latency which could be crippling to the behavior algorithm. Run on a fast machine though, OMNI works flawlessly and is the perfect example of a simple computer vision robot in action.

Integrated System

OMNI's software is built upon TA's Mike Pridgen and Thomas Vermeer's PVR board, powered by an Atmel processor. This board is responsible for all servo control, sensor reading and behavior algorithms. This board proved to be extremely versatile and was able to handle polling my 8 infrared sensors simultaneously quite admirably.

Bump sensors were simply attached to a pull up resistor and then to the digital inputs on the PVR board, while the IR sensors were connected to the AD inputs pins. I also attached an xBee wireless serial communication module to the RS-232 communication pins on the board.

OMNI also has an LCD screen, which was mostly used for debugging purposes. This is attached and controlled through the provided LCD control header on the PVR board.



System Connectivity Diagram

Mobile Platform

OMNI's chassis is constructed out of a two 8-inch wide octagonal platforms sandwiched by 2 inch steel standoffs. The servos controlling the omni-directional wheels are mounted on the lower platform along with the board power supply. All of the electronics and sensors are mounted on the top platform. These platforms were cut using the T-tech in lab out of high-quality wood and then sanded. Many coats of metallic red spray paint were then applied to give OMNI some sex appeal.

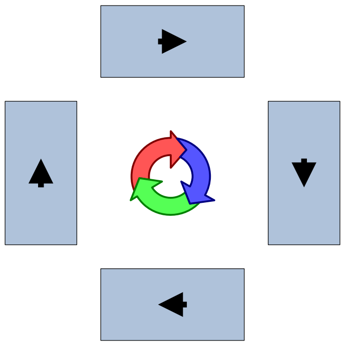
My main design decision when creating OMNI was to make the chassis as symmetrical as possible. In some cases this proved to be out of necessity, for the wheels and the IR and bump sensors, but I also chose to mount the IP webcam in the center of the top platform on top of a 360 degree servo allowing for a full range of view in all directions, though it was not utilized in the final behavior algorithm.

The omni-directional wheels (Kornylak Transwheels) provide for motion along 2 axes, which allowed OMNI to move in any direction without turning. This was a huge advantage for designing the behavior algorithms later on down the road. I had heard reports of these omni-directional wheels having poor grip, so I made sure to order the special variety which included a rubberized coating on the cross-axis inset wheels. These provided more than adequate traction. I also went with the dual layer omni-wheels, despite the significant price difference. This was a good decision, since the single layer wheels have reports of not allowing for smooth cross-axis motion.

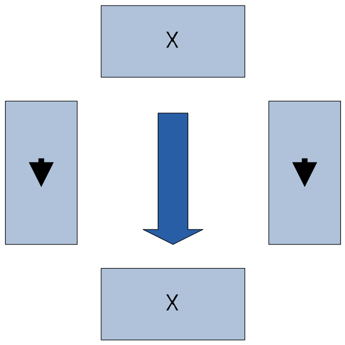
Kornylak Transwheel (dual layer)

Actuation

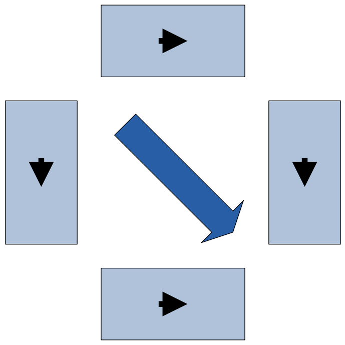
Actuation is where I reaped the rewards of the simplicity of my mobile platform's design. The algorithms to control the omni-directional wheels were very simple to calculate, as illustrated here, keeping in mind that the magnitude of rotation for any single servo can be altered to produce intermediary results.



Rotate



Forward



Diagonal

Sensors

Bump Switches

Description

Bump switches are the simplest of OMNI's sensors. There are four 2 terminal microswitches mounted underneath the top panel of the chassis. These are staggered such that each is responsible for sensing collisions on a quarter of OMNI's periphery. Each switch has a flexible metal rod welded to it to extend the action for the switch around the edge of OMNI's chassis.

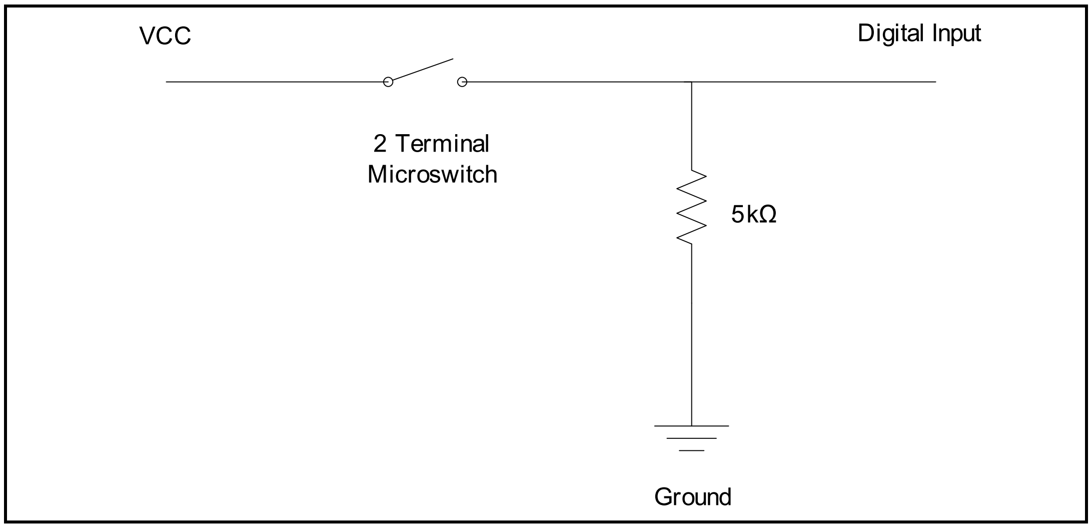
These switches are intended to be a last resort for obstacle avoidance, allowing OMNI to detect when its IR sensor bank has failed to catch an obstacle and on which side that a collision has occurred. With this information OMNI can then attempt to navigate away from the collision, orient itself for better IR detection of the object which it just collided with, and then continue on its way.

In OMNI's code these switches are handled through interrupts which fire whenever a switch goes high. There are external interrupts configured to listen on each of the digital input ports which are connected to a bump switch. When the switch goes logic high, the interrupt fires and starts the subroutine for a bump switch collision. The alternative to this method was polling the switches continuously in the program loop, but the use of interrupts helps to save some time in main program loop allowing it to execute faster.

Connection

These sensors are really nothing more than simple SPST (single pole, single throw) switches. This means that when the switch is activated it makes a connection between the two terminals, otherwise there is no connection. This is not very conducive to digital circuits, as I want the switch to pull down to logic zero (ground) when it is not activated, rather than hang floating. In order to rectify this I created circuits for each switch which tied the logic terminal through a large resistor (>5kΩ) to ground. When a connection is made and VCC is allowed to pass through the switch to the logic terminal, very little current and voltage is lost over the high resistance branch. In retrospect it would've been easier to use 3 terminal microswitches.

Wiring Diagram:



Sensor Data

This sensor only returns values of logic high and logic low, corresponding to whether the switch is activated or deactivated respectively.

Switch Position	Value at Digital Input
Activated	1
Deactivated	0

Infrared Distance Sensors

Description

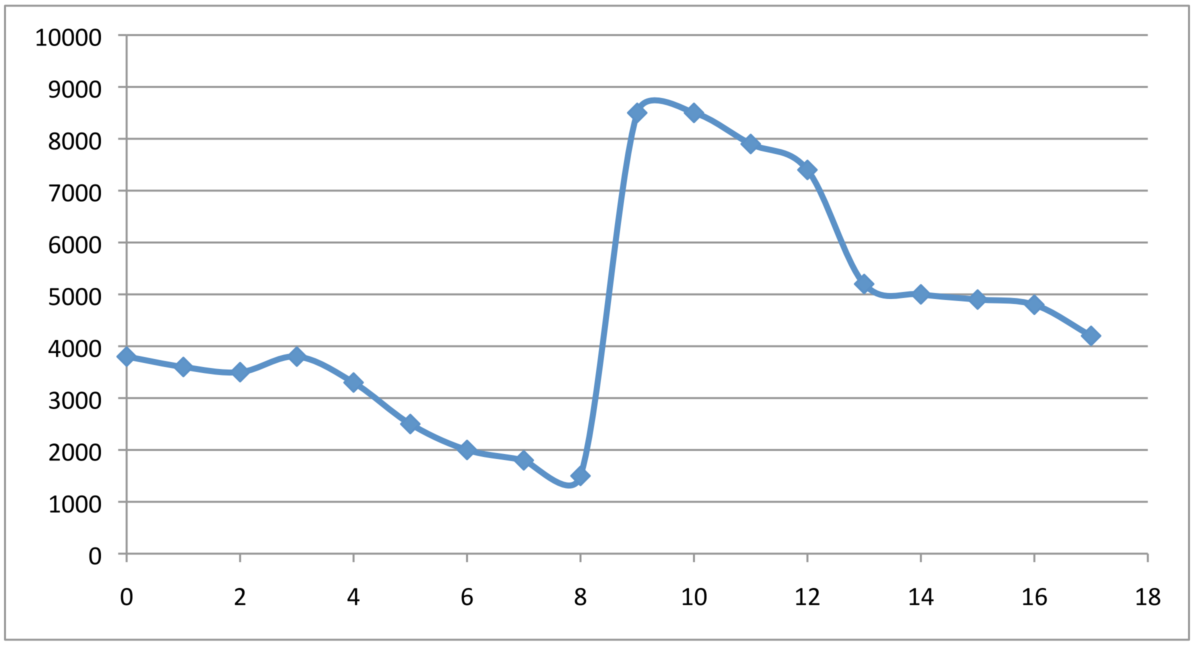
OMNI employs a ring of eight Sharp GP2D120 distance measuring sensors along each edge of the octagonal chassis. Since the length of each of these edges is approximately three inches, and the sensor itself takes up more than an inch, OMNI is able to effectively judge its distance from its surroundings in all directions with almost no blind spots. These sensors claim to be able to determine distances within the range of 4 to 30cm, but I was really only able to measure a little over half that range accurately as the sensor data later in this report shows. Due to the minimum range requirement of 4 inches, I set the sensors about an inch back from the edge of the chassis so that the range of acceptable values fell closer to OMNI's proximity. Since there is about another 2 to 3 inches of wheels and bump switches, this puts the minimum sensor reading just at just before bump switches, perfect for obstacle avoidance and wall-following applications.

The biggest issues that I encountered with these sensors are that they are extremely noisy and they return a rather unusual non-linear function for the distance. The first issue was solved programmatically using some filtering techniques which kept a short history of previous values, reducing the effect that IR sensor noise has on the servo control algorithm. This significantly reduced some jittering behaviors that were emerging due to OMNI acting on sensor noise. The issue with the non-linear function for distance is somewhat of a non-issue since OMNI does not need exact distance measurements to perform obstacle avoidance, it simply reacts when an object passes a certain distance threshold. This threshold was determined experimentally, and fortunately it occurs only once in the distance function curve so I did not have to worry about throwing out sensor readings which are too close or too far returning the same value.

Connection

Each infrared sensor is connected to Port A on the PVR board, the designated ADC port. There are just enough (8) lines on this port for all of my sensors. The IR sensors require both a power and ground input and output an analog voltage. These sensors were made to connect with a 3-pin JST connector, pictured below. This cable was modified to connect directly into the 3 header pins on Port A.

Sensor Data



This graph shows the IR sensor readings at increasing distance in inches from the IR sensor. Values shown are the average of several tests on several individual sensors. As you can see the data is anything but linear. The range that I am concerned with is the 3 – 5 inch range, where OMNI is about to collide with an incoming object. This range reads about 3000 – 3500, and this is set to the threshold for my ADC values to trigger obstacle avoidance.

These IR sensors also tend to idle at a value in the low hundreds, around 300 or so, when they are first switched on. Once an object is brought within 18 inches or so of the sensor they 'wake up' and start sensing the values shown above. After this happens they tend to idle at around 4000. I'm not sure why the sensors have this peculiarity, but it does not affect my algorithm.

There is also a huge spike in the values at around 9 inches away from the sensor. The values then decay from that point on and eventually hover around 4000, the idling value.

Wireless IP Camera

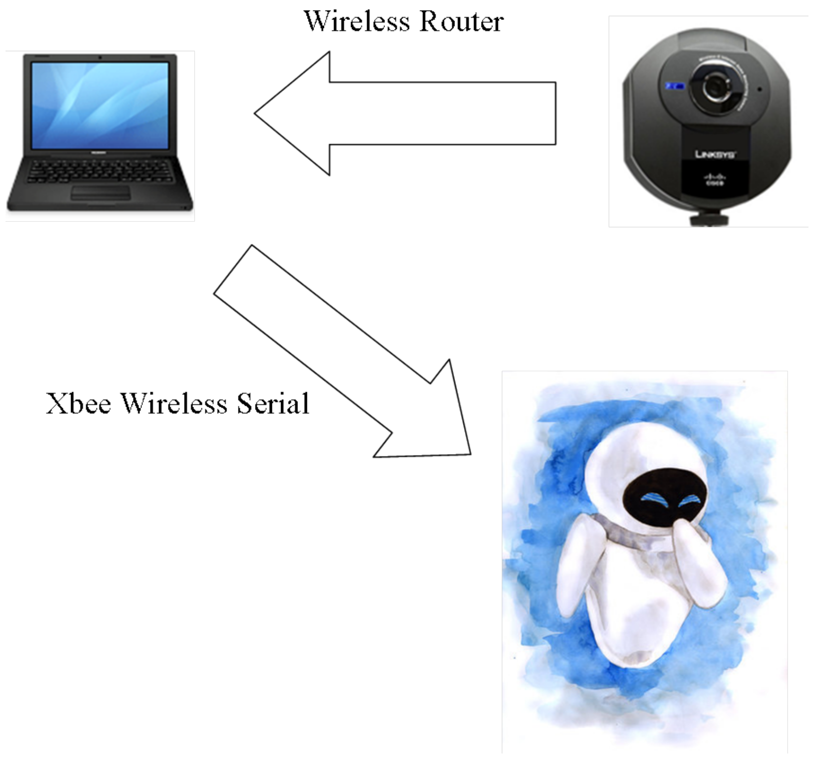
Description

OMNI is equipped with one Linksys WVC54GCA wireless IP camera which is used to acquire and transmit visual data to a remote machine for processing. This camera is mounted on a hacked GWS S03NXF 2BB servo for a full 360 degree field of view around OMNI. OMNI uses this camera to pick out a target color in its environment and home in on the source of the color. Obstacle avoidance is handled by a ring of eight infrared range finders and four bump switches mounted on the periphery of OMNI's chassis, the camera is intended solely for color tracking in its current incarnation.

In this way OMNI is able to balance obstacle avoidance with behavioral objectives. OMNI gives obstacle avoidance a much higher priority than behavioral objectives so that obstacles will always be avoided, keeping any vision behaviors from interfering when a collision is imminent. Future plans for development include incorporating the camera into the obstacle avoidance algorithm.

Connection

OMNI's camera is not connected directly to the PVR board, it relays streaming video through a wireless router to a computer. The Linksys WVC54GCA wireless IP camera is configured to connect to a specified wireless router and supply a MJPEG stream at a sub-address under its DHCP address. The video stream is accessed on the computer and it is piped through vision algorithms for color detection and location. Once the approximate location of the highest area of concentration of a specified color is found, coordinates are relayed back to OMNI over a pair of linked Xbee wireless serial communication modules. These coordinates are then be used to modify the variables driving each of the wheels, allowing OMNI to attempt to move closer to the colored object. The webcam is connected to its own power supply on OMNI which supplies 5V.



The streaming video from the webcam must be parsed and analyzed for key aspects, in this case color. OMNI is a color tracking robot, so color analysis is the primary function of the vision algorithms.

The first thing the vision algorithms tackle is dilating the colors of the picture. This algorithm enhances the colors of the foreground while obscuring the colors of the background. This is accomplished by taking each pixel that appears to exist in a blob (a large area of similarly colored pixels) and then expanding the blob by coloring all adjacent pixels the same color as the blob. By doing this large blob objects become accentuated, such as the object that OMNI is tracking, while insignificant background noise or smaller color concentrations are eroded and eventually eliminated by the expansion of their neighbors.

After this is completed the next step is to pick out the particular color that is being tracked. This is done through color filtering. The algorithm searches the image for pixels which are within a certain threshold (10 RGB units for my algorithm) from specified values, and turns all pixels that do not fall within this threshold black. The result is an image with just the object and a black background, assuming that there weren't any other significantly large objects in the picture that were a similar color.

Now that the algorithm has highlighted the object in the image all that is left is to determine its location. To do this all of the x and y coordinates of the non-black pixels in the image are averaged to determine the average x and y coordinate of the colored pixels, which usually is located at a decent estimation of the center of the object being tracked. A box is then drawn around this coordinate, the size of which is determined by the deviation of the pixels coordinates. For example the box is small and well-fitted when the previous steps in the vision algorithm are able to filter out most of the noise in the image.

Image algorithms are implemented using the RoboRealm computer vision toolkit, an incredibly useful computer vision resource.

Behaviors

OMNI’s overall software system can be broken down into two modules: obstacle avoidance and colored object tracking. For obstacle avoidance OMNI utilizes its ring of infrared rangefinder sensors to determine if there are any objects within a set distance from OMNI in every direction. OMNI also has 4 radial bump switches which trigger should OMNI collide with an obstacle. OMNI uses the measurements that it reads off of these sensors to gauge and weight its servo control algorithms. OMNI also uses a wireless IP webcam to track a colored object. The tracking itself is done remotely, as described in the previous sections, but once the vision algorithm calculates the location of the object within the frame it transmits a command via wireless serial communication telling OMNI to weight its servo control variables to either rotate left or right to center the object or to move straight because the object is already centered. Keep in mind that these are simply weights, so any other algorithms such as obstacle avoidance and previous tracking weights will still be taken into account when calculating the final servo control value. This allows for a very dynamic control mechanism for OMNI’s movement, allowing it to move away from an obstacle while tracking an object at the same time and also gives it more realistic appearing motion as behavior modifications to servo values tend to “meld” into a smoother motion. It should also be noted that all servo values decay at a constant rate so that when no stimulus is applied to any sensors OMNI will remain stationary or, in a later version of the behavior software, will enter a “seek” mode in which it rotates looking for an object to track. For better understanding of OMNI’s behaviors, please see the code attached in the appendix.

Experimental Layout and Results

Sensor testing was the first experimentation that I performed, testing the IR ranges and the bump switch actuation. These experiments can be seen in my ‘Sensors’ section.

I also experimented quite a bit with the best algorithms for color tracking, also enumerated in the ‘Sensors’ section.

Later experimentation involved mainly the camera tracking, as I was still trying to get OMNI to operate using the 360 degree servo to rotate the wireless IP camera. These experiments ultimately led me to temporarily abandon the use of this servo since it became apparent that I would need a motor encoder or some form of sophisticated vision waypoint system to determine the exact orientation of the camera at any point in time. Thus, due to time constraints this was not implemented in my final build.

As an interesting side note, I was very pleased with the layout of my IR sensors. The octagonal layout allowed for 3 sensors to always be detecting any extended wall that OMNI approached. Because of this, OMNI surprised me by being remarkably adept at wall-following simply using its obstacle avoidance routine.

Conclusion

Although I did not manage to get OMNI working as smoothly as I had hoped due to processor limitations on my laptop and time constraints, it was still a rewarding project that I will continue to tinker with for some time to come. I started with big plans and I'm pleased that while I was not able to implement some of them OMNI at least has the hardware framework down for me to implement these plans later on down the road. I would like to make OMNI's camera rotate on its own axis and incorporate this ability into its behaviors. I think that allowing it to look in any direction while also moving in any direction (due to the omni-directional wheels), is a very cool combo with lots of potential for cool behaviors.

Overall, the project cost upwards of around $600 which was about $300 more than I expected. I didn't fully appreciate how expensive certain parts would be. For example, the IP webcam turned out to be an item that goes for >$100, I got mine refurbished for $60. The battery and charger for my camera came to $40. The IR sensors were $7 each and were expensive since I had to order a new batch after finding that my original ones were too long-range and would fail to sense objects close to OMNI.

If I had more time I would've tracked down a faster laptop to run my software on so that there would be minimal latency. Also, I would've implemented a more complex computer vision routine. My original plan was to have OMNI recognize glyphs with the camera and then reproduce them with a marker attached to a push servo on its underbelly, going through the bottom platform. While this is technically possible, I would've wanted to invest some money in some high precision motors to run my wheels rather than the relatively unreliable servos that I have.

Overall, I have learned a lot in this class. Building the robot was definitely the trickiest bit for me, since I'm more of a software guy. The first revision is never perfect, and I plan on improving on OMNI's design in the future.

Documentation

Sources for information outside my own personal testing and development:

- Mike and Thomas' PVR board documentation and C libraries
- Digi X-CTU (for xBee configuration and drivers)
- AVR Studio, WinAVR
- Roborealm computer vision toolkit
- Documentation for:
    - Sharp IR Sensors
    - WVC wireless IP webcam
    - All servos
    - xBee module

Appendices

Appendix A:
Behavior, C code

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



/*! Define that selects the Usart used in example. */
#define USART USARTF0


#define IR_THRESH_HIGH    3500
#define IR_THRESH_LOW     3000
#define SERVO_STEP                25
#define CAMERA_STEP               15
#define CAMERA_DECAY      1
#define DECAY                     5
#define SERVO_ZERO                10
#define CAMERA_SERVO_ZERO 5
#define OS_TO                     100

int main(void)
{

        /* Variable used to send and receive data. */
        //uint8_t sendData;
        uint8_t receivedData;

        /* This PORT setting is only valid to USARTC0 if other USARTs is used a
         * different PORT and/or pins is used. */
        /* PIN3 (TXD0) as output. */
        PORTF.DIRSET = PIN3_bm;

        /* PC2 (RXD0) as input. */
        PORTF.DIRCLR = PIN2_bm;

        /* USARTC0, 8 Data bits, No Parity, 1 Stop bit. */
        USART_Format_Set(&USART, USART_CHSIZE_8BIT_gc, USART_PMODE_DISABLED_gc,
false);

        /* Set Baudrate to 9600 bps:
         * Use the default I/O clock fequency that is 32 MHz.
         * Do not use the baudrate scale factor
         *
         * Baudrate select = (1/(16*(((I/O clock frequency)/Baudrate)-1)
         *                 = 208
         */
        // xmegaInit() sets internal clock to 32MHz, this must be adjusted accordingly
        // this was fun to figure out
        USART_Baudrate_Set(&USART, 208 , 0);

        /* Enable both RX and TX. */
        USART_Rx_Enable(&USART);
        USART_Tx_Enable(&USART);
```

```
	xmegaInit();								//setup
XMega
	delayInit();								//setup
delay functions

	ServoCInit();								// init
servo ports
	//ServoDInit();
	PORTH_DIR |= 0x00;							// set
port H to inputs


	ADCAInit();
	// init ADC for IR
	lcdInit();
	lcdString("OMNI Operational");

	int C0,C1,C2,C3,D0,D1,D2,D3;
	// initial servo values
	C0 = 0;
	C1 = 0;
	C2 = 0;
	C3 = 0;
	D0 = 0;
	D1 = 0;
	D2 = 0;
	D3 = 0;
	// variables for handling overshoot
	char prev = ' ';
	int os_timeout = 0;

	while(true) {


		int timeout = 1000;
		// check for serial communication from computer
		do{
		/* Wait until data received or a timeout.*/
		timeout--;
		}while(!USART_IsRXComplete(&USART) && timeout != 0);


		// if it didnt time out, read the character
			if (timeout !=0) {
				receivedData = USART_GetChar(&USART);
			}
			// locked onto tracked object, damp camera servo control
			if ((char)receivedData == 's') {
				// move straight
				C2 -= CAMERA_STEP;
```

```
                C3 += CAMERA_STEP;
                C1 -= CAMERA_STEP;
                C0 += CAMERA_STEP;
                /*
                if (D0 > 0) {
                        D0 -= CAMERA_DECAY * 15;
                        if (D0 < 0) {
                                D0 = 0;
                        }
                }
                else if (D0 < 0) {
                        D0 += CAMERA_DECAY * 15;
                        if (D0 > 0) {
                                D0 = 0;
                        }
                }
                */
        }


        /*
        if ((char)receivedData == 'b') {
                // spin
                C0 += CAMERA_STEP;
                C1 += CAMERA_STEP;
                C2 += CAMERA_STEP;
                C3 += CAMERA_STEP;

                os_timeout--;
                if (prev == 'l') {
                        D0 -= CAMERA_STEP * 5;
                }
                else if (prev == 'r') {
                        D0 += CAMERA_STEP * 5;
                }

        }

        */


// turn right
if((char) receivedData == 'r') {

        // turn right
        C0 -= CAMERA_STEP;
        C1 -= CAMERA_STEP;
        C2 -= CAMERA_STEP;
        C3 -= CAMERA_STEP;

}
// turn left
else if((char) receivedData == 'l') {

        // turn left
        C2 += CAMERA_STEP;
        C0 += CAMERA_STEP;
```

```
                C3 += CAMERA_STEP;
                C1 += CAMERA_STEP;
        }


        // move camera
        // functionality disabled, need a redesign to implement camera
turning on own axis
        //ServoD0((int)D0);

        // servo control update, kill signal if value is less than SERVO_ZERO
to prevent drifting
        /*
        if ((D0 <= CAMERA_SERVO_ZERO) && (D0 >= -CAMERA_SERVO_ZERO)) {
                TCD0_CCA = 0;
        }
        else {
                ServoD0(D0);
        }
        */



        if ((C0 <= SERVO_ZERO) && (C0 >= -SERVO_ZERO)) {
                TCC0_CCA = 0;
        }
        else {
                ServoC0(C0);
        }
        if ((C1 <= SERVO_ZERO) && (C1 >= -SERVO_ZERO)) {
                TCC0_CCB = 0;
        }
        else {
                ServoC1(C1);
        }
        if ((C2 <= SERVO_ZERO) && (C2 >= -SERVO_ZERO)) {
                TCC0_CCC = 0;
        }
        else {
                ServoC2(C2);
        }
        if ((C3 <= SERVO_ZERO) && (C3 >= -SERVO_ZERO)) {
                TCC0_CCD = 0;
        }
        else {
                ServoC3(C3);
        }

        // decay servo control magnitudes
        if (C0 > 0) {
                C0 -= DECAY;
        }
```

```
else if (C0 < 0){
         C0 += DECAY;
}
if (C1 > 0)  {
         C1 -= DECAY;
}
else if (C1 < 0){
         C1 += DECAY;
}
if (C2 > 0)  {
         C2 -= DECAY;
}
else if (C2 < 0){
         C2 += DECAY;
}
if (C3 > 0)  {
         C3 -= DECAY;
}
else if (C3 < 0){
         C3 += DECAY;
}

// bump sensor check
switch (PORTH_IN) {
         case 8:
                  C0 -= SERVO_STEP;
                  C2 += SERVO_STEP;
                  break;
         case 4:
                  C0 += SERVO_STEP;
                  C2 -= SERVO_STEP;
                  break;
         case 64:
                  C1 -= SERVO_STEP;
                  C3 += SERVO_STEP;
                  break;
         case 128:
                  C1 += SERVO_STEP;
                  C3 -= SERVO_STEP;
                  break;
}

// check IR thresholds and update servos accordingly
if((ADCA0() < IR_THRESH_HIGH) && (ADCA0() > IR_THRESH_LOW)) {
         C1 -= SERVO_STEP;
         C2 += SERVO_STEP;
         C3 += SERVO_STEP;
```

```
        C0 -= SERVO_STEP;
}
if((ADCA1() < IR_THRESH_HIGH) && (ADCA1() > IR_THRESH_LOW)) {
        C1 -= SERVO_STEP;
        C3 += SERVO_STEP;
}

if((ADCA2() < IR_THRESH_HIGH) && (ADCA2() > IR_THRESH_LOW)) {
        C2 -= SERVO_STEP;
        C3 += SERVO_STEP;
        C1 -= SERVO_STEP;
        C0 += SERVO_STEP;
}
if((ADCA3() < IR_THRESH_HIGH) && (ADCA3() > IR_THRESH_LOW)) {
        C0 += SERVO_STEP;
        C2 -= SERVO_STEP;
}
if((ADCA4() < IR_THRESH_HIGH) && (ADCA4() > IR_THRESH_LOW)) {
        C3 -= SERVO_STEP;
        C0 += SERVO_STEP;
        C1 += SERVO_STEP;
        C2 -= SERVO_STEP;
}
if((ADCA5() < IR_THRESH_HIGH) && (ADCA5() > IR_THRESH_LOW)) {
        C1 += SERVO_STEP;
        C3 -= SERVO_STEP;
}
if((ADCA6() < IR_THRESH_HIGH) && (ADCA6() > IR_THRESH_LOW)) {
        C1 += SERVO_STEP;
        C0 -= SERVO_STEP;
        C2 += SERVO_STEP;
        C3 -= SERVO_STEP;
}
if((ADCA7() < IR_THRESH_HIGH) && (ADCA7() > IR_THRESH_LOW)) {
        C0 -= SERVO_STEP;
        C2 += SERVO_STEP;
}


// cap servo speeds
if (D0 > 15) {
        D0 = 15;
}
else if (D0 < -15) {
        D0 = -15;
}
if (C0 > 100) {
```

```
                        C0 = 100;
                }
                else if (C0 < -100) {
                        C0 = -100;
                }
                if (C1 > 100) {
                        C1 = 100;
                }
                else if (C1 < -100) {
                        C1 = -100;
                }
                if (C2 > 100) {
                        C2 = 100;
                }
                else if (C2 < -100) {
                        C2 = -100;
                }
                if (C3 > 100) {
                        C3 = 100;
                }
                else if (C3 < -100) {
                        C3 = -100;
                }


        }

        /* Disable both RX and TX. */
        USART_Rx_Disable(&USART);
        USART_Tx_Disable(&USART);

}
```

Appendix B

Vision Algorithm Code – Roborealm XML

```xml
<head><version>2.18.5</version></head>
<Read_HTTP>
  <is_active>TRUE</is_active>
  <url>http://192.168.1.3:80/img/mjpeg.cgi</url>
  <password>jester44</password>
  <method_index>1</method_index>
  <username>admin</username>
</Read_HTTP>
<Rotate>
  <mode>-180</mode>
  <custom_degree>0.0</custom_degree>
</Rotate>
<Mean disabled>
  <filter_size>20</filter_size>
</Mean>
<Segment_Colors>
  <tolerance>80</tolerance>
  <detail_size>20</detail_size>
</Segment_Colors>
<Color_Filter>
  <colors>#A9DDB4

#90D5A8

</colors>
  <adjust_lighting>FALSE</adjust_lighting>
  <min_distance>5</min_distance>
  <min_value>100</min_value>
</Color_Filter>
<Center_of_Gravity>
  <show_coord>TRUE</show_coord>
  <display_as_annotation>FALSE</display_as_annotation>
  <color_index>2</color_index>
  <connect_line>FALSE</connect_line>
  <size_index>5</size_index>
  <use_subpixel>FALSE</use_subpixel>
  <density>1</density>
  <show_box>TRUE</show_box>
  <box_size>9</box_size>
  <overlay_image>Current</overlay_image>
  <shape_index>1</shape_index>
  <show_cog>TRUE</show_cog>
</Center_of_Gravity>
<If_Statement>
  <comparison_type_1>3</comparison_type_1>
```

```
    <comparison_1>5</comparison_1>
    <join_1>1</join_1>
    <value_1>200</value_1>
    <variable_1>COG_X</variable_1>
    <comparison_4>-1</comparison_4>
    <comparison_type_4>-1</comparison_type_4>
    <join_2>-1</join_2>
    <value_2>0</value_2>
    <comparison_type_3>-1</comparison_type_3>
    <variable_2>COG_X</variable_2>
    <comparison_3>-1</comparison_3>
    <join_3>-1</join_3>
    <comparison_type_2>3</comparison_type_2>
    <comparison_2>3</comparison_2>
    <has_else>FALSE</has_else>
  </If_Statement>
  <Serial>
    <enable_send_sequence>TRUE</enable_send_sequence>
    <port>COM3 - USB Serial Port</port>
    <data_bits>7</data_bits>
    <baud>6</baud>
    <flow_control_in_x>FALSE</flow_control_in_x>
    <read_rate_index>-1</read_rate_index>
    <flow_control_dsr>FALSE</flow_control_dsr>
    <flow_control_cts>FALSE</flow_control_cts>
    <send>1</send>
    <send_only_on_change>FALSE</send_only_on_change>
    <flow_control_out_x>FALSE</flow_control_out_x>
  </Serial>
  <end_if/>
  <If_Statement>
    <comparison_type_1>3</comparison_type_1>
    <comparison_1>3</comparison_1>
    <join_1>-1</join_1>
    <value_1>400</value_1>
    <variable_1>COG_X</variable_1>
    <comparison_4>-1</comparison_4>
    <comparison_type_4>-1</comparison_type_4>
    <join_2>-1</join_2>
    <comparison_type_3>-1</comparison_type_3>
    <comparison_3>-1</comparison_3>
    <join_3>-1</join_3>
    <comparison_type_2>-1</comparison_type_2>
    <comparison_2>-1</comparison_2>
    <has_else>FALSE</has_else>
  </If_Statement>
  <Serial>
```

```
    <enable_send_sequence>TRUE</enable_send_sequence>
    <port>COM3 - USB Serial Port</port>
    <data_bits>7</data_bits>
    <baud>6</baud>
    <flow_control_in_x>FALSE</flow_control_in_x>
    <read_rate_index>-1</read_rate_index>
    <flow_control_dsr>FALSE</flow_control_dsr>
    <flow_control_cts>FALSE</flow_control_cts>
    <send>r</send>
    <send_only_on_change>FALSE</send_only_on_change>
    <flow_control_out_x>FALSE</flow_control_out_x>
  </Serial>
  <end_if/>
  <If_Statement>
    <comparison_type_1>3</comparison_type_1>
    <comparison_1>5</comparison_1>
    <join_1>1</join_1>
    <value_1>400</value_1>
    <variable_1>COG_X</variable_1>
    <comparison_4>-1</comparison_4>
    <comparison_type_4>-1</comparison_type_4>
    <value_2>200</value_2>
    <comparison_type_3>3</comparison_type_3>
    <variable_2>COG_X</variable_2>
    <comparison_3>1</comparison_3>
    <join_3>-1</join_3>
    <value_3>0</value_3>
    <comparison_type_2>3</comparison_type_2>
    <variable_3>COG_X</variable_3>
    <comparison_2>3</comparison_2>
    <has_else>FALSE</has_else>
  </If_Statement>
  <Serial>
    <enable_send_sequence>TRUE</enable_send_sequence>
    <port>COM3 - USB Serial Port</port>
    <data_bits>7</data_bits>
    <baud>6</baud>
    <flow_control_in_x>FALSE</flow_control_in_x>
    <read_rate_index>-1</read_rate_index>
    <flow_control_dsr>FALSE</flow_control_dsr>
    <flow_control_cts>FALSE</flow_control_cts>
    <send>s</send>
    <send_only_on_change>FALSE</send_only_on_change>
    <flow_control_out_x>FALSE</flow_control_out_x>
  </Serial>
  <end_if/>
  <If_Statement>
```

```
    <comparison_type_1>3</comparison_type_1>
    <comparison_1>1</comparison_1>
    <value_1>0</value_1>
    <variable_1>COG_X</variable_1>
    <comparison_4>-1</comparison_4>
    <comparison_type_4>-1</comparison_type_4>
    <join_2>-1</join_2>
    <value_2>630</value_2>
    <comparison_type_3>-1</comparison_type_3>
    <variable_2>COG_X</variable_2>
    <comparison_3>-1</comparison_3>
    <join_3>-1</join_3>
    <comparison_type_2>3</comparison_type_2>
    <comparison_2>4</comparison_2>
    <has_else>FALSE</has_else>
  </If_Statement>
  <Serial>
    <enable_send_sequence>TRUE</enable_send_sequence>
    <port>COM3 - USB Serial Port</port>
    <data_bits>7</data_bits>
    <baud>6</baud>
    <flow_control_in_x>FALSE</flow_control_in_x>
    <read_rate_index>-1</read_rate_index>
    <flow_control_dsr>FALSE</flow_control_dsr>
    <flow_control_cts>FALSE</flow_control_cts>
    <send>b</send>
    <send_only_on_change>FALSE</send_only_on_change>
    <flow_control_out_x>FALSE</flow_control_out_x>
  </Serial>
  <end_if/>
```