

Pino

Robot Report 3 **Final Written Report**

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Student name: Julian Ramlal

Robot name: Pino

Date: 7/7/2007

Instructors name(s): Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

TA name(s): Adam Barnett
Kevin Claycomb

Table of Contents

1. Abstract.....	3
2. Introduction.....	4
3. Integrated System.....	5
3.1. System Description.....	5
3.2. Theory of Operation	5
3.3. System Layout	6
3.3.1. Sensors	6
3.3.2. Motors	6
3.3.3. Motor/Power Board	6
3.3.3.1. Battery Monitor.....	6
3.3.3.2. Optical Isolators	7
3.3.3.3. Motor Driver	7
4. Mobile Platform	9
5. Actuation.....	10
5.1. Thrust Fans	10
5.2. Lift Fan	10
6. Sensors	11
6.1. Sonar	11
6.2. Infrared	12
6.3. Optical Mouse Sensor.....	12
6.4. Compass.....	12
6.5. Integration.....	13
7. Behaviors	14
7.1. Collision Avoidance	14
7.2. Sprint.....	14
7.3. 360 Turn.....	14
7.4. Algorithms	15
8. Experimental Layout and Results	16
8.1. Sensors.....	16
8.1.1. Infrared.....	16
8.1.2. Sonar	16
8.1.3. Digital Compass.....	17
8.1.4. Optical Mouse.....	18
9. Conclusion	19
10. Documentation.....	20
11. Appendices.....	21
11.1. Code Snippets	21
11.1.1. Infrared.....	21
11.1.2. Sonar	21
11.1.3. Motor Control	22
11.1.4. LCD.....	22
11.1.5. Optical Mouse.....	23

1. Abstract

Pino is an autonomous ACV (Air Cushion Vehicle) based off a toy R/C hovercraft. It glides over a thin frictionless film of air and propels itself by using fans mounted at the rear of the craft. Pino's purpose is to demonstrate that a robot can perform collision avoidance even when working with a frictionless surface with drift. Additionally, Pino will demonstrate special maneuvering required for navigating a frictionless surface such as 360 degree turning and reversing. Pino's main behavior is a full sprint to a dead stop where the ACV propels itself forward at maximum speed only to stop inches away from any stationary obstacle in front.

2. Introduction

Hovercrafts succumb to the challenges inherent in moving along a frictionless surface much like a hockey puck moving across ice. This presents an interesting navigational problem as even when the hovercraft itself turns it may still be drifting in another direction so it is up to the microcontroller to perform the necessary actions to compensate for that drift so that the hovercraft can turn and maneuver as well as or better than a typical speed boat.

Typical recreational hovercrafts are able to perform 360 degree turning by shifting their weight as well as turning the hovercraft quickly in the direction. With enough skill the pilot will be able to turn the craft completely about and eventually start going in the other direction. As this is not a trivial maneuver it presents an interesting challenge to include this maneuver in the design of a robot hovercraft.

This robot project, named Pino, is able to overcome the navigational challenges of ACVs (Air Cushion Vehicles) and use its newfound frictionless freedom to perform non-trivial maneuvers without colliding with objects in its environment.

This paper attempts to fully detail Pino's abilities and behaviors and what it uses to accomplish the aforementioned tasks/behaviors. Pay close attention to the limitations of the sensor information provided to the robot and the software techniques used to compensate. Near the end of the paper is a comprehensive section detailing experiment results from the Pino project, these are of particular importance as they detail the success and capabilities of the finished project.

3. Integrated System

3.1. System Description

The robot is organized to be a completely reactive system whereby the microcontroller, Mavric IIB with ATmega128 [1], collects data from the sensors and

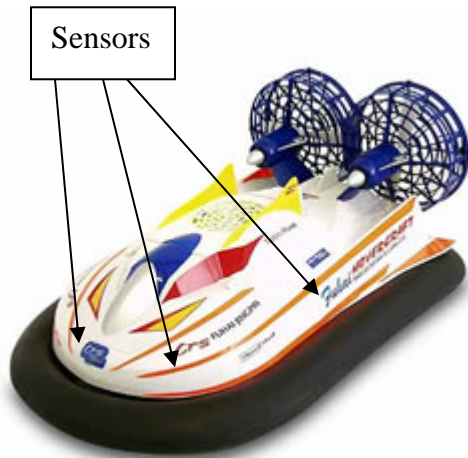


Figure 1. Peripheral sensor locations on the toy hovercraft upon which the robot will be based.

below.

3.2. Theory of Operation

One of the more important aspects of the system integration is the operation of the hovercraft itself. Basically, a hovercraft glides on a thin film of air and propels itself using the two fans in the back. Navigating the hovercraft entails reducing the power of a fan on either side of the craft to make it go in the opposite direction and reversing the thrust altogether to slow the craft down.

One of the biggest problems with navigating is that frictionless thin film of air that causes the craft to drift while turning. To improve system navigation, the robot is fitted with a compass and optical mouse sensor that will indicate the current heading of the craft which it will use to compensate for the discrepancy between the craft

processes it into signals to be sent to the motors. Avoidance sensors are mounted on the periphery of the vehicle (Figure 1.) and include a digital compass that, in combination with the IR and sonar, cooperates to perform collision avoidance. In addition to these avoidance sensors is an optical mouse sensor that is used to supplement the sensor data from the other three sensors. After sampling the sensor data, the microcontroller will send signals to the motors so as to control the speed and heading of the vehicle as well as the direction of the sweeping sonar. This flow of information and control can be seen quite clearly in Figure 2.

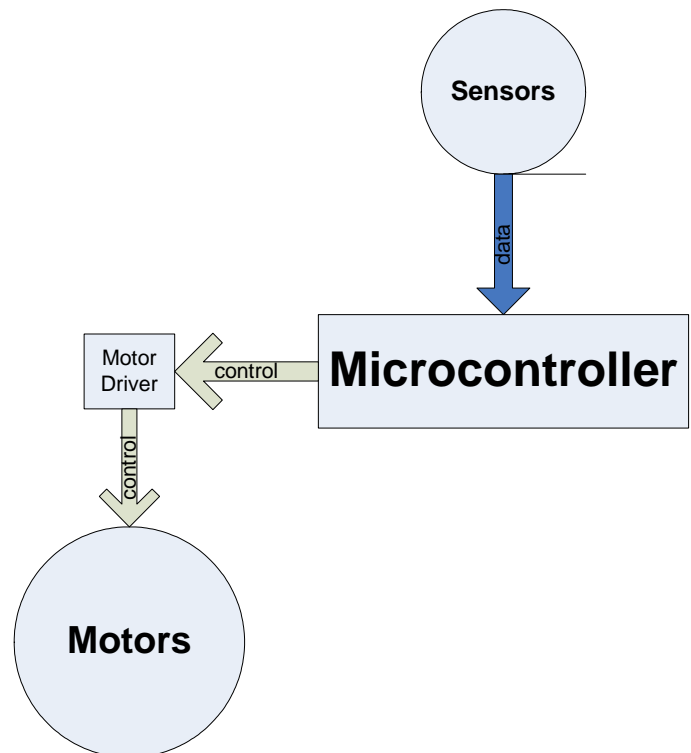


Figure 2. Flow diagram for data and control within the system.

heading and the direction of motion. This feature, combined with the side IR sensors and front sonar will allow the craft ample collision avoidance at high speeds.

The robots' primary behavior is collision avoidance whereby the robot attempts to avoid obstacles while even traveling at high speeds. Its secondary behavior is the 360 degree turn that the robot performs when avoiding obstacles and when sprinting. The sprint maneuver is apart of the secondary behavior profile and involves the robot moving at a high speed and stopping on a dime using the 360 turn.

When considering how the robot will complete its goals and behaviors in the scheme of the system structure it is important to realize that they are the central driving forces behind the robots' logic. The sensors feed the microcontroller all the data it needs so that the behaviors will be able to come up with control instructions to supply the motor driver and although the structure is relatively simple, it is this constant flow of information and control that allows the robot to function.

3.3. System Layout

3.3.1. Sensors

Pino's avoidance sensor profile includes two front-facing medium-ranged sonar, two side-facing wide-angled medium-ranged infrared sensors, a central digital compass and an optical mouse sensor located beneath the vehicle. This sensor layout, Figure 3, also shows the sensor ranges and overlaps of the directional sensor. The optical mouse sensor is placed beneath the hovercraft and is fitted to the base resulting in a small amount of drag on the hovercraft. This effect is negligible especially in comparison with the benefits.

3.3.2. Motors

Similarly, a close look at the motor system in Figure 4. reveals two similar divisions with only one stepper motor in the exploratory division. That motor is used to sweep the sonar and PIR sensor array 360 degrees around the robot. The remaining motors include three 12V DC Axial fans and one 12V DC Impeller [2]. The robot uses the axial fans for steering and propulsion and the impeller, or centrifugal fan, is used to provide lift, i.e. fill the small chamber of air beneath the craft. The reason for using a centrifugal fan that they are more capable of producing higher air pressures than axial fans.

3.3.3. Motor/Power Board

The motors require a large amount of current and, therefore, cannot be run on the same battery supply as the electronics. To route power through the system we had to use a high amperage n-channel MOSFET [3] and interface the two battery supplies via optical isolators.

3.3.3.1. Battery Monitor

We used a LM393 low-voltage comparator [4] to check the voltage of the batteries to see if they drop below a certain level. Pino uses Lithium Polymer batteries, which can be volatile when over discharged. As a

measure of sanity a small amount of hysteresis was also included into the circuitry so that the light does not flicker as much when the voltage reaches the danger point [5] [6] [7]. The change to the comparator was a minor one an included only adding a positive feedback resistor between the output and the input reference voltage, allowing for a buffer zone between high and low comparator readings.

3.3.3.2. Optical Isolators

H11B1 Motorola optical isolators with Darlington [8] [9] output were used to communicate between the two different battery planes and to ensure a constant +5V input to the isolators as well as the comparator and motor driver, we added a voltage regulator to the circuit, the LM7805 1A +5V voltage regulator [10]. The Darlington output is a bit slow but for our purposes they will serve quite well as our motors won't be doing that much switching.

3.3.3.3. Motor Drivers

To drive our motors we decided on the SN754410NE dual half H-bridge chips from Texas Instruments (TI). The relatively simple chip design only required us to sent it a signal and it would provide the motor with the necessary power to move.

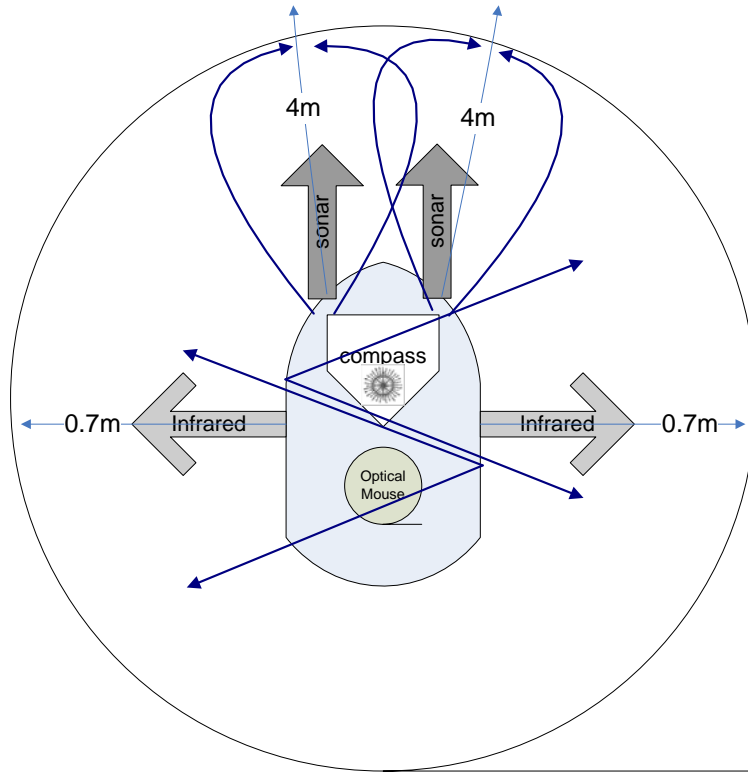


Figure 3. Sensor array layout with ranges and detection zones. The front sonar do overlap so as to have three detection zones: left, middle and right.

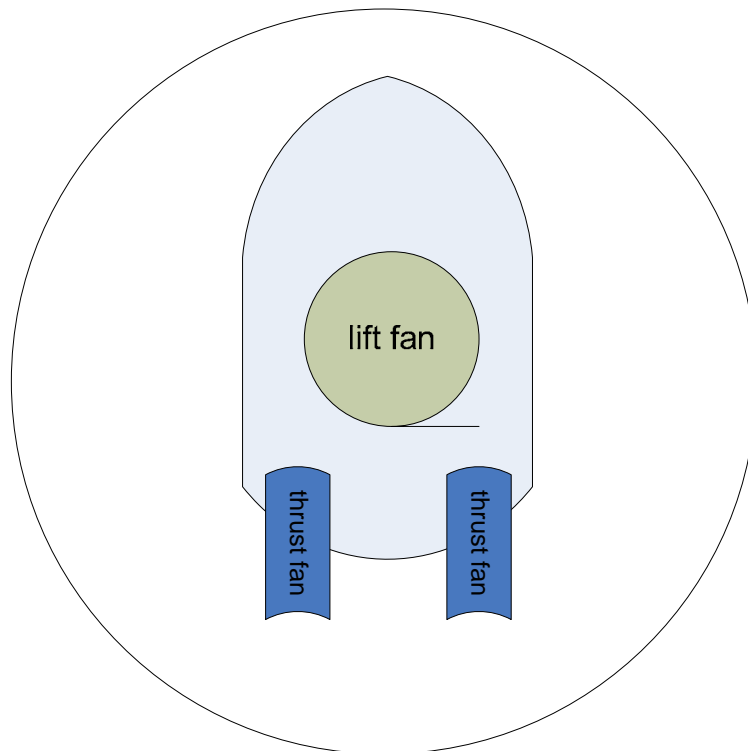


Figure 4. Motor layout on the hovercraft. The thrust fans are directed behind the craft.

4. Mobile Platform

The robots platform is constructed in the likeness of a typical hovercraft and will look something like the shape in Figure 5. with the Mavric IIB board mounted on the front and

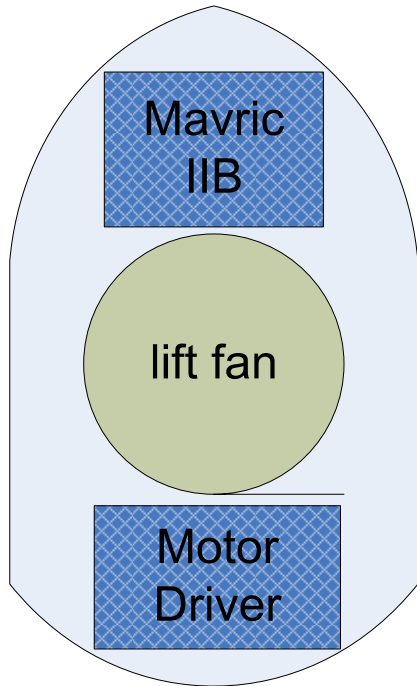


Figure 5. Platform layout with the shaded areas indicating printed circuit boards.

the motor driver board on the back. The batteries are typically inside the craft on the platform on either side of the lift fan or wherever else they need to be placed in order to balance out the weight of the craft so that the center of gravity is in the middle of the lift fan. Our aim with placing the batteries is so that no side of the hovercraft is closer to the ground than the other since this could cause undue drift. In addition to increasing stability, the purpose of this mounting is also to place all electromagnetic equipment as far away from the compass as possible in order to reduce interference and prevent any damage to the sensor.

With this simple platform design the robot maintains the likeness of a hovercraft as well as the stability garnered from its wide base of approximately 6". The lift fan has a height of approximately 2" resulting in a platform height of around 2.5". The height has little effect on stability given that it is within a



reasonable range. The length adds to the stability of the craft since it increases the area of the air pocket beneath the craft. All these factors come together to form the platforms dimensions, 11.5"x6"x2.5".



Figure 6. Skirt attachment frame that is fixed to the platform base.

The platform is cut out of 1/8" Birchwood, a relatively lightweight and sturdy material. Mounted on the top of this base are the Mavric IIB, the lift fan and the motor driver board. Mounted on the bottom of the platform are two plastic frames, Figure 6, that are used for attaching the skirt to the platform. These pieces are also the main channels through which air leaves the vehicle and enters the skirt and the air pocket below.

5. Actuation

Actuation takes place through three motors, all of which are fans. Two fans mounted on the rear robot platform to provide thrust and steering for the hovercraft. One fan mounted at the craft's center of gravity and is used to provide lift. The rear fans are driven by a custom motor driver board that contains several motor drivers and the lift fan is powered by a high-current MOSFET.

5.1. Thrust Fans

The forward thrust fan is one 12V DC AFB1212SHE PC fan from Delta, 120mm x 38mm, that runs at 1050mA and outputs 151CFM and the rear thrust fans consist of two 12V DC EFB1212VHE PC fans from Delta, 120mm x 38mm, that run at 800mA and output 142CFM. Both are axial fans capable of producing an air pressure of about 0.5 in H₂O but, due to their high axial airflow, are better off used as propulsion.



5.2. Lift Fan

The lift fan is a 12V DC Diplomat - DD402112K1R Impeller from Comair Rotron [2], 101.6mm x 55.9mm, that runs at 2750mA and outputs 115CFM. This is a centrifugal fan capable of producing an air pressure of about 2.010 in H₂O. Since this fan can operate at a greater air pressure it is perfect for use as a lift fan.



6. Sensors

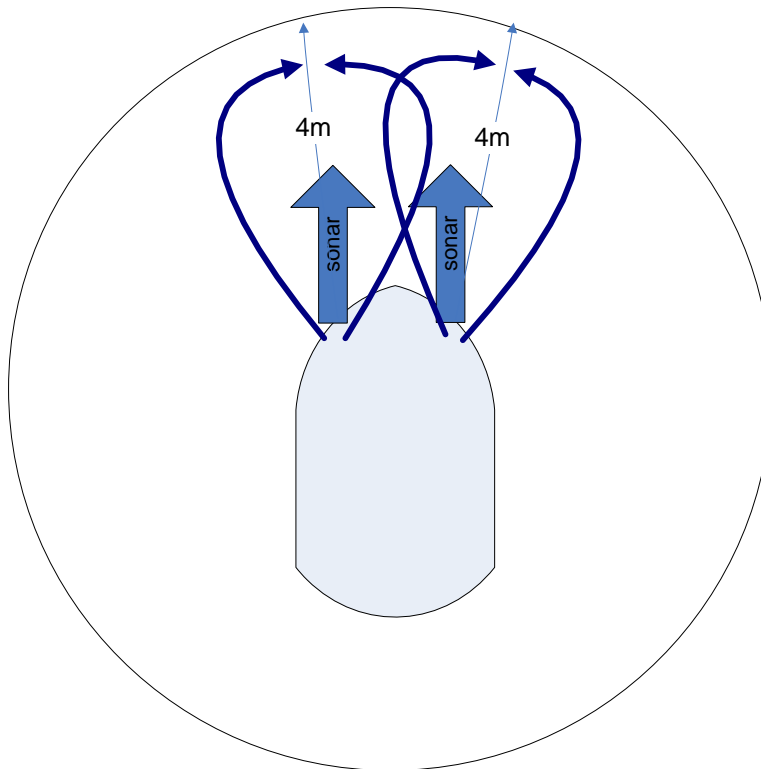
The robot is fitted with a complete array of avoidance sensors including sonar, infrared, a digital compass and a optical mouse sensor. Avoidance sensors provide the microcontroller with data that it can use to avoid obstacles. The avoidance sensors can be found around the periphery of the craft

6.1. Sonar

The first type of sonar is the Devantech SRF05 (left) medium range sonar (0.03m to 4m) that operates at 40kHz on 5V and outputs range via an analog signal that must be timed by the microcontroller for an accurate reading. That reading, taken with the speed of sound, is converted to distance. These sonar are mounted on the front of the craft and provide the robot with full range obstacle detection in front.



The second type of sonar is the Devantech SRF02 (right) long range sonar (0.15m to 6m) that operates at 40kHz on 5V at



4mA and outputs range digitally over I2C. This sonar is mounted on the back of the vehicle and is used to aid the craft in making 360 degree turns and any maneuver that requires the craft to move backwards. This sonar has a very long range and a very fast refresh rate, approximately one ranging every 75ms, allowing for speed measurement, the secondary purpose of this sensor.

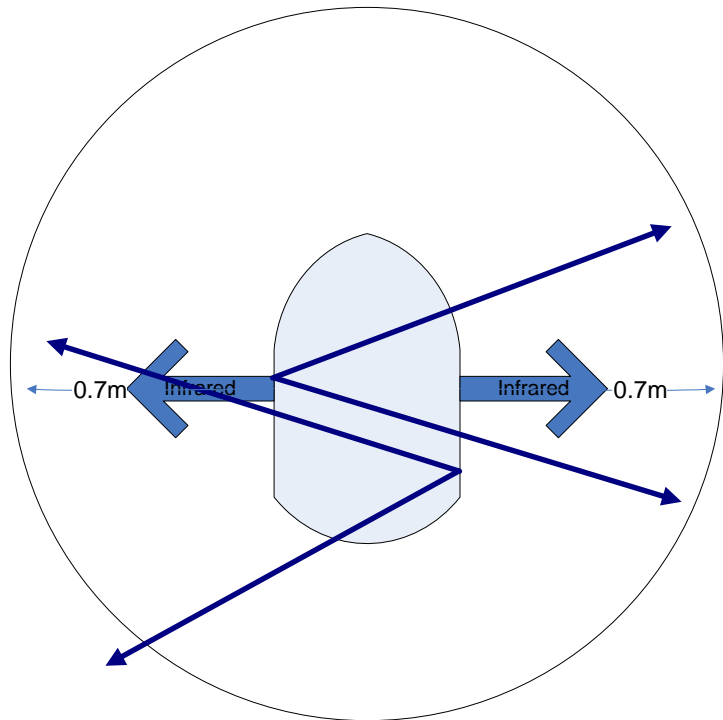
6.2. Infrared

These infrared ranging sensors are Sharp



GP2Y0A21YK
short range
(10cm to 80cm),
wide angle

distance measuring sensors that run at 5V on 30mA with a response time for 40ms and output in analog. These sensors are mounted on the sides of the craft and provide the robot with wide angle object ranging on the sides and work together with the front sonar for obstacle avoidance. One important thing to note about these sensors and the mounting is that they are cross-mounted, which means that they face across from each other, effectively reducing the already short range from 10cm – 80cm to 0cm – 70cm.



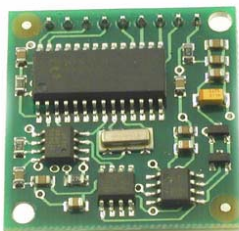
6.3. Optical Mouse Sensor

The optical mouse sensor was obtained from a General Electric optical mouse. This sensor requires direct contact with the ground in order to work thus making the data somewhat unreliable. Nevertheless, with supplementary information from the other sensors we can filter out extraneous data. This sensor provides its information serially over a data line in the form of X and Y coordinates. All the optical mouse sensor information, including surface quality (SQUAL) values and the surface image are stored inside registers on the sensor.



6.4. Compass

This sensor is a Devantech R117-COMPASS compass and it uses a magnetic field sensor to detect the Earth's magnetic field resulting in a digital reading that indicates the



direction of the vehicle. It works at 5V on 20mA with 0.1 degree resolution and 3-4 degree accuracy and a digital output over I2C. The compass is used to determine the direction of the vehicle so that it can compensate for any discrepancy between the vehicles direction and its heading, the direction it is moving in, and is considered part of the robots' obstacle avoidance sensor package. It is mounted in the middle of the craft, as far

away from any magnetic interference as possible so that the reading is not skewed while the robot is active and so that any strong magnetic fields do not damage the sensor itself as the magnetic field sensor is very sensitive to even the slightest magnetic fields.

6.5. Integration

All the sensors send their data directly to the microcontroller where all the behaviors are sorted out. The sensor data drives behaviors and these behaviors are arbitrated within the microcontroller resulting in motor control signals. Typically there are only two of these kinds of signals since the craft only has to modulate the speed of the rear fans in order to navigate. Primary sensor information comes from the infrared and sonar sensors with supplementary data from the digital compass and optical mouse sensor. These secondary sensors are used to verify the primary sensor data as well as attempt to provide information such as speed, direction and heading. The optical mouse sensor is responsible for speed and direction while the digital compass takes care of heading.

7. Behaviors

7.1. Collision Avoidance

Collision avoidance behavior is as simple as it sounds – the microcontroller processes the data from the avoidance sensor package and uses it to detect and avoid obstacles. Within that package is sensor data from the digital compass and that is used as temperance on the ranging sensors such that the robot is able to know that it still needs to speed up or slow down even though the ranging sensors might say differently. For example, once an obstacle is detected the craft could make a 360 degree turn but still be drifting in the same direction. At this point the ranging sensors would no longer detect an obstacle but by using the craft direction, provided by the compass, the controller will know that it still needs to speed up in order to slow down. Speed and direction data provided by the optical mouse sensor will also help to guide the craft but will be more of a supplement as it is less reliable.

7.2. Sprint

While in the spring behavior, the craft will move forward at full speed and make a full stop only inches from the nearest stationary obstacle. This behavior is coupled with the 360 Turn behavior that it uses to perform the full stop. Consequently, the craft will be facing in the opposite direction when it reaches the obstacle.

7.3. 360 Turn

This behavior requires the craft to be moving and have ample room in front in order for it to succeed. Essentially, the craft will shut down one of its propulsion fans, causing it to drift into a turn while still moving in the previous direction. After completing the turn, the craft will be facing the opposite direction and at this point the craft can either continue drifting for backwards movement or begin moving forward for reverse/braking.

7.2. Algorithms

Pino uses a form a fuzzy logic to define its behavior and data representation. Sensor data is collected by the system then divided out into categories for distance and speed.

IRL	IRR		<u>Distance</u>	
	VF	F	C	VC
VF	TR	TR	TL	TL
F	TR	TR	TL	TL
C	TL	TL	S	S
VC	TL	TL	S	S

Table 1. Fuzzy Logic Table that is used to generate the algorithm that drives the robot. The table is laid out in the form of a K-Map with each cell being an action recommendation. robot should take.

Distance and speed result from sonar and infrared readings and translate into either Very Far, Far, Close or Very Close [VF, F, C, VC] for distance and Very Fast, Fast, Slow and Very Slow [VF, F, S, VS] for speed. Each fuzzy value is represented by a binary value from 00 to 11 for use when calculating logic equations from the fuzzy logic tables. The fuzzy logic table in Table 1 is used for the distance calculations for the IR sensors during the turning behavior.

Values	Results
VF = 00	TL = Turn Left
F = 01	TR = Turn Right
C = 11	F = Forward
VC = 10	S = Stop

Table 2. The binary representations for the fuzzy logic values are displayed on the left and the action recommendations or results are displayed on the right.

Using the binary values for the fuzzy logic values in the table we can derive equations for each action recommendation (TR = Turn Right, TL = Turn Left, F = Forward, S = Stop). For example, from Table 1 above we can derive logic equations dependent directly on the sensor results.

$$TR = \sim IRL1 \ \&\& \ \sim IRR1$$

$$TL = IRL1 \ \&\& \ \sim IRR1 \ \text{or} \ \sim IRL1 \ \&\& \ IRR1$$

$$S = IRL1 \ \&\& \ IRR1$$

Consequently, with this and several other logic tables we can derive upwards of twenty logic equations to define the behavior of the system. From there the robot just carries out the action recommendations directly by turning, accelerating or stopping.

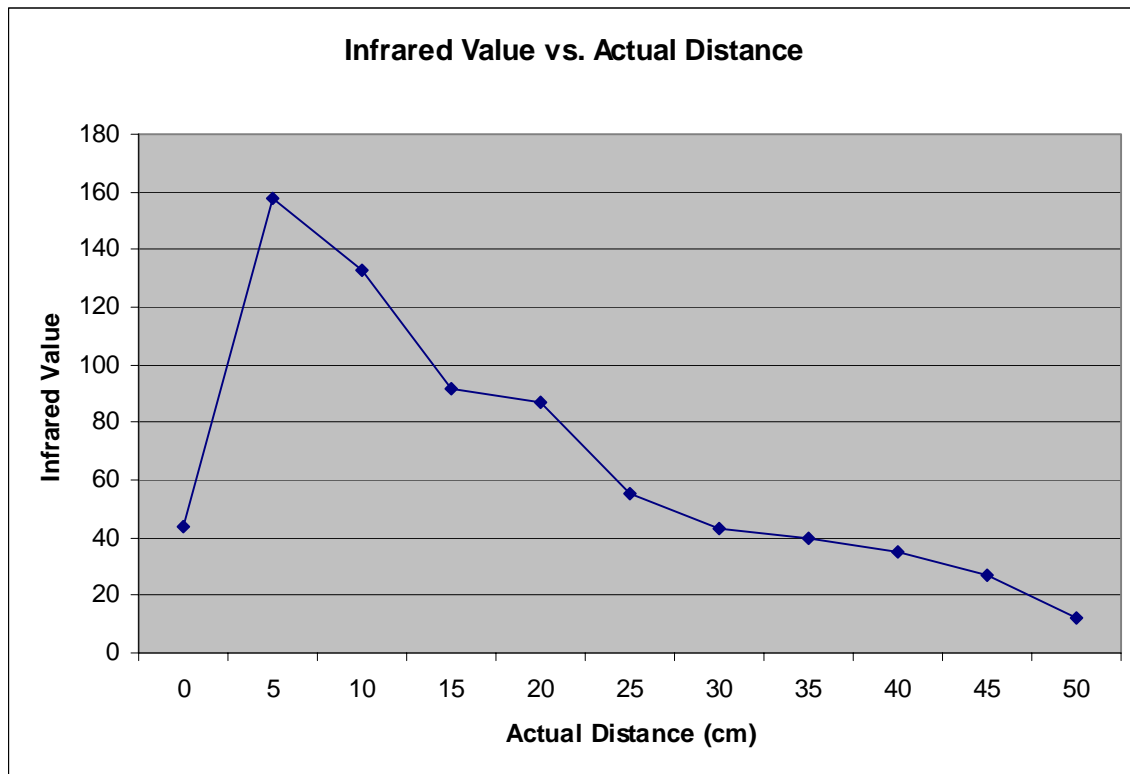
8. Experimental Layout and Results

8.1. Sensors

All sensors were experimentally laid out against an accurate metric in order to gauge what range of values they would produce during operation.

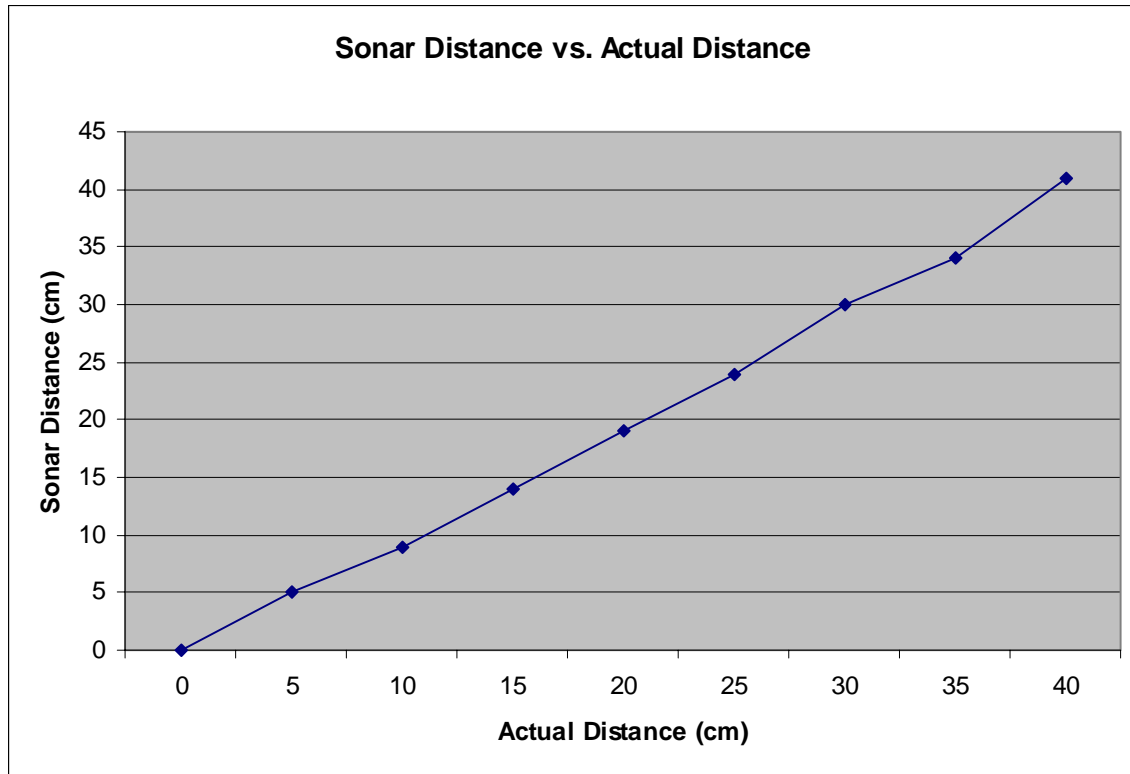
8.1.1. Infrared

The IR sensor experiment consisted of one IR sensor lined up against a tape measure. A flat non-reflective relatively dense sheet of paper was moved towards the sensor in increments of 10cm. The sensor readings were recorded and are plotted on the graph below.



8.1.2. Sonar

The Sonar underwent the same treatment as the IR whereby the sensor was sensor lined up against a tape measure. A flat non-reflective relatively dense sheet of paper was moved towards the sensor in increments of 10cm. The sensor readings were recorded and are plotted on the graph below. The programmed equation used to convert the actual sensor reading to the recorded values was mathematically derived using basic physics equations – $\text{distance_cm} = (\text{SensorValue}/58) - 26$. The reason we are dividing the sensor value by 58 is because the sensor value indicates the time in microseconds between when the sonar echo was sent and the time it was received by the sensor. We then subtract 26 to bias the readings so that 0cm indicates a position directly in front of the sensor.



8.1.3. Digital Compass

Experimentally testing the digital compass was a fairly simple ordeal and involved simply moving the sensor, while keeping it as flat as possible, in a clockwise fashion over a real compass. The sensor values were then read off and compared to the actual values. Any noted deviation from the actual value can be contributed to either human error or a lack of calibration of the sensor. Nevertheless, the compass does not need to find north, only the difference in values between readings is significant.

#	Actual Degree	Compass Value	Compass Degree	Difference
1	0	11	1.1	1.1
2	45	455	45.5	0.5
3	90	911	91.1	1.1
4	135	1352	135.2	0.2
5	180	1783	178.3	1.7
6	225	2219	221.9	3.1
7	270	2683	268.3	1.7
			Standard Deviation	1.3428571

8.1.4. Optical Mouse

The optical mouse was similarly straightforward, the sensor was moved across a suitable surface in strict X and Y directions. Any deviations in the readings indicate strictly human error and are apparent in the oscillations in the readings since the device registers the change in position from the last reading. Experimental comparison with actual movement values was unnecessary as the sensor would primarily be used to determine the direction of movement and not so much the distance.

Mouse Sensor Movement

#	Left (X, Y)		Right (X, Y)		Up (X, Y)		Down (X, Y)	
	Left X	Left Y	Right X	Right Y	Up X	Up Y	Down X	Down Y
1	50	5	136	127	0	15	133	147
2	72	4	134	0	127	20	127	159
3	78	145	133	1	127	24	5	157
4	68	129	135	128	129	10	0	175
5	66	133	135	2	128	13	3	167
6	55	135	134	3	0	22	135	138
7	60	140	137	0	130	28	5	131
8	62	143	142	127	127	37	128	144
9	61	0	129	0	129	25	128	138
10	66	129	131	0	135	26	128	141

Interfacing between the Mavric IIB and the optical mouse sensor was a challenge since the sensor works on an open collector interface. To send the mouse a low signal we could output low like normal but to output a high level we had to set the port pin to input and activate it's internal pull-up resistors. This interface actually works quite nicely since the communication is bi-directional and the host can take charge of the communication channel at any time. All the information that I used to implement this interface was gathered from internet sites detailing the PS/2 mouse protocol [11] [12]. A small sample of the important code for the optical mouse interface is included in the appendix section.

9. Conclusion

This was a challenging project due in part to the high level of engineering that goes into building these vehicles. Although their design seems simple it is by no means trivial. A lot of thought and hard work went into designing and balancing these vehicles so that they can float even with the weight of the battery. In the end Pino was fully functional and able to move around autonomously. The movement, however, was a bit sluggish as modifying the craft introduced indeterminate flaws that impaired its movement. The modified version did not inflate its skirt or create nearly as much lift as the original and even the rear fans seemed a bit sluggish. Nevertheless, the craft was eventually able to glide across a frictionless surface of air and navigate by controlling the two rear fans. The movement was a bit sloppy but that is to be expected most humans have trouble completely controlling a hovercraft.

In retrospect, the design could have been improved significantly and if I ever had the chance to work on a similar project again it surely would turn out a much better product than the result of this project.

10. Documentation

- [1] Atmel Corporation. "8-Bit AVR Microcontroller with 128K Bytes In-System Programmable Flash ATmega128," October 2006, http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf
- [2] COMAIR ROTRON, "Diplomat DD4021 Series 100mm Brushless DC Motorized Impeller," May 2007, <http://www.comairrotron.com/images/drawings/newpdf/DD4021.pdf>
- [3] SGS-THOMSON MICROELECTRONICS, "N - CHANNEL ENHANCEMENT MODE POWER MOS TRANSISTORS," September 1996, <https://www.jameco.com/Jameco/Products/ProdDS/210526.pdf>
- [4] National Semiconductor, "LM393 - Low Power Low Offset Voltage Dual Comparators," August 2002, <http://cache.national.com/ds/LM/LM193.pdf>
- [5] R. Moghimi, "Curing Comparator Instability with Hysteresis," December, 2000, <http://www.analog.com/library/analogDialogue/archives/34-07/comparators/>
- [6] R. Paisley, "Voltage Comparator Information And Circuits," June 01, 2007, <http://home.cogeco.ca/~rpaisley4/Comparators.html>
- [7] D. V. Ess, "Comparator Hysteresis in a Nutshell," March 2007, <http://www.analogzone.com/acqt0327.pdf>
- [8] Electus Distribution, "OPTOCOUPPLERS – WHEN AND HOW TO USE THEM," August 2001, http://www1.jaycar.com.au/images_uploaded/optocoup.pdf
- [9] Motorola Semiconductor, "6-PIN DIP Optoisolators Darlington Output (Low Input Current)," May 1995, <http://www.ortodoxism.ro/datasheets/motorola/H11B3.pdf>
- [10] FAIRCHILD SEMICONDUCTOR, "LM7805 - 3-Terminal 1A Positive Voltage Regulator," June 2007, <http://www.fairchildsemi.com/pf/LM/LM7805.html>
- [11] T. Engdahl and G. Ries, "Mouse (PS/2) pinout and signals," June 25, 2005, http://pinouts.ru/Inputs/PS2Mouse_pinout.shtml
- [12] A. Chapweske, "The PS/2 Mouse/Keyboard Protocol," May 9, 2003, <http://www.computer-engineering.org/ps2protocol/>
- [13] BDMICRO, "Sample Source Code for the MAVRIC / MAVRIC-II Mega128 Board," August 2007, <http://www.bdmicro.com/code/>

11. Appendices

11.1. Code Snippets

A fair amount of code can be contributed to outside sources such as the I2C interface utility and the digital compass software [13]. Other code that concerns the behaviors and the other sensors are completely custom written.

11.1.1. Infrared

```
void initIR() {
    // Enable ADC, Free Running, with frequency of 250kHz
    ADCSRA = _BV(ADEN) | _BV(ADSC) | _BV(ADIE) | _BV(ADPS2) |
_BV(ADPS1) | _BV(ADPS0);

    // Left Adjusted ADC and Channel 0
    ADMUX = _BV(ADLAR) | _BV(REFS0);
}

// ADMUX is used to multiplex the ADC between IR1 & IR2
void capture_ir1() {
    ADMUX = _BV(ADLAR) | _BV(REFS0);
    ADCSRA |= _BV(ADSC);
}

void capture_ir2() {
    ADMUX = _BV(ADLAR) | _BV(REFS0) | _BV(MUX0);
    ADCSRA |= _BV(ADSC);
}

SIGNAL(SIG_ADC) {
    ir = ADCL;
    ir = ADCH;
}
```

11.1.2. Sonar

```
void initSonarSRF05(void) {
    SONAR1DDR |= SONAR1IO;
    SONAR2DDR |= SONAR2IO;

    TIFR      |= _BV(ICF1);
    TCCR1B    |= _BV(CS11); /* CTC, prescale = 8 */
    TCNT1     = 0;
    TIMSK     |= _BV(TICIE1);
    SONAR1    |= SONAR1POWER;
}

void capture_sonar1() {
    ACSR &= ~_BV(ACIC);
    TIFR  |= _BV(ICF1);

    SONAR1 ^= SONAR1TP;
    ms_sleep(1);
    SONAR1 ^= SONAR1TP;
}
```

```

    TCNT1 = 0;
}

SIGNAL(SIG_INPUT_CAPTURE1) {
    sonar1_ping = ICR1;
    sonar1_ping = (sonar1_ping/58)-26;
}

```

11.1.3. Motor Control

```

void initPWM() {
    MOTORSDDR |= MOTORIO;

    TCCR3A      |= _BV(COM3B1) | _BV(COM3B0) | _BV(COM3C1) |
    _BV(COM3C0) | _BV(WGM31) | _BV(WGM30);
    TCCR3B      |= _BV(WGM33) | _BV(WGM32) |
    _BV(CS31); // TOP in ICR1 and prescale = 8
    OCR3A       = 200;
    OCR3B       = 100;
    OCR3C       = 50;
    TCNT3       = 0;
}

```

11.1.4. LCD

```

void writeLCD_D(unsigned char d) {
    static unsigned char send;

    send = d & 0xF0;

    LCD = LCD_DATA_H | send;
    LCD = LCD_DATA_L | send;
    us_sleep(LCD_DELAY);

    d &= 0x0F;
    d <<= 4;

    LCD = LCD_DATA_H | d;
    LCD = LCD_DATA_L | d;
    us_sleep(LCD_DELAY);
}

```

```

void writeLCD_C(unsigned char d) {
    static unsigned char send;

    send = d & 0xF0;

    LCD = LCD_COMMAND_H | send;
    LCD = LCD_COMMAND_L | send;
    ms_sleep(LCD_DELAY);

    d &= 0x0F;
    d <<= 4;

    LCD = LCD_COMMAND_H | d;
    LCD = LCD_COMMAND_L | d;
}

```

```

    ms_sleep(LCD_DELAY);
}

```

11.1.5. Optical Mouse

```

void initOptical(void) {
    OPTICAL_DDR |= OPTICAL_IO;

    OPTICAL |= OPTICAL_POWER;

    // IDLE
    SET(OPTICAL, OPTICAL_DATA);           // float data high
    SET(OPTICAL, OPTICAL_CLOCK);         // float clock high

    UNSET(OPTICAL_DDR, OPTICAL_CLOCK);    // clock is input
    UNSET(OPTICAL_DDR, OPTICAL_DATA);     // data is input
}

/** Open Collector Operation:
 *
 * High - DDRx.y = 0, PORTx.y = 1
 * Low   - DDRx.y = 1, PORTx.y = 0
 */

/**
 * Device States:
 *
 * 1. Idle: CLK line and DATA line are floating high.
 * In this mode, the mouse may start transmitting data
 * at any time.
 *
 * 2. Inhibit: DATA line is floating high, CLK line is drawn
 * low by the host to prevent the mouse from initiating any
 * transmission. The mouse will buffer its data until the
 * line is Idle again.
 *
 * 3. Request to send: CLK line is floating high, host draws
 * DATA line low so that the mouse will prepare to receive
 * a command packet.
 */

/**
 * Pre:      Device State = IDLE
 * Post:     Device State = INHIBIT
 */
void mouseRecieveByte(void) {
    uint8_t parity = 0;
    m_rcv = 0;

    // INHIBIT
    SET(OPTICAL_DDR, OPTICAL_CLOCK);      // clock is output
    UNSET(OPTICAL_DDR, OPTICAL_DATA);     // data is input

    SET(OPTICAL, OPTICAL_DATA);           // float data high
    UNSET(OPTICAL, OPTICAL_CLOCK);       // pull clock low, float data high
}

```

```

// IDLE
SET(OPTICAL, OPTICAL_DATA);           // float data high
SET(OPTICAL, OPTICAL_CLOCK);         // float clock high

UNSET(OPTICAL_DDR, OPTICAL_CLOCK);    // clock is input
UNSET(OPTICAL_DDR, OPTICAL_DATA);     // data is input

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT0);
    parity++;
}

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT1);
    parity++;
}

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT2);
    parity++;
}

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT3);
    parity++;
}

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT4);
    parity++;
}

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT5);
    parity++;
}

```



```

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT6);
    parity++;
}

waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();
if ( recieve & OPTICAL_DATA ) {
    SET(m_rcv, BIT7);
    parity++;
}

waitUntilClockLow();
parity = OPTICAL & OPTICAL_DATA;
waitUntilClockHigh();

// stop bit
waitUntilClockLow();
recieve = OPTICAL;
waitUntilClockHigh();

// INHIBIT
SET(OPTICAL_DDR, OPTICAL_CLOCK);           // clock is output
UNSET(OPTICAL_DDR, OPTICAL_DATA);         // data is input

SET(OPTICAL, OPTICAL_DATA);               // float data high
UNSET(OPTICAL, OPTICAL_CLOCK);           // pull clock low
}

/**
 * Pre:      Device State = INHIBIT
 * Post:     Device State = INHIBIT
 */
void mouseSendByte(uint8_t m_snd) {
    uint8_t parity = 1;
    OPTICAL_DDR = OPTICAL_IO;
    OPTICAL = 0x04;

    // INHIBIT
    SET(OPTICAL_DDR, OPTICAL_CLOCK);       // clock is output
    UNSET(OPTICAL_DDR, OPTICAL_DATA);     // data is input

    SET(OPTICAL, OPTICAL_DATA);           // float data high
    UNSET(OPTICAL, OPTICAL_CLOCK);       // pull clock low

    // sleep for 150us
    us_sleep(15);

    // Host Request to Send
    UNSET(OPTICAL_DDR, OPTICAL_CLOCK);    // clock is input

```

```

SET(OPTICAL_DDR, OPTICAL_DATA); // data is output

UNSET(OPTICAL, OPTICAL_DATA); // float clock high, pull data low
SET(OPTICAL, OPTICAL_CLOCK); // float clock high

if ( m_snd & BIT0 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;

    while(1) {
        tmp = OPTICAL_PIN & 0x0f;

        if ( tmp != 4 )
            break;
    }
}
else
    UNSET(OPTICAL, OPTICAL_DATA);

waitUntilClockHigh();

waitUntilClockLow();
if ( m_snd & BIT1 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;
}
else
    UNSET(OPTICAL, OPTICAL_DATA);
waitUntilClockHigh();

waitUntilClockLow();
if ( m_snd & BIT2 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;
}
else
    UNSET(OPTICAL, OPTICAL_DATA);
waitUntilClockHigh();

waitUntilClockLow();
if ( m_snd & BIT3 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;
}
else
    UNSET(OPTICAL, OPTICAL_DATA);
waitUntilClockHigh();

waitUntilClockLow();
if ( m_snd & BIT4 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;
}
else
    UNSET(OPTICAL, OPTICAL_DATA);

```

```

waitUntilClockHigh();

waitUntilClockLow();
if ( m_snd & BIT5 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;
}
else
    UNSET(OPTICAL, OPTICAL_DATA);
waitUntilClockHigh();

waitUntilClockLow();
if ( m_snd & BIT6 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;
}
else
    UNSET(OPTICAL, OPTICAL_DATA);
waitUntilClockHigh();

waitUntilClockLow();
if ( m_snd & BIT7 ) {
    SET(OPTICAL, OPTICAL_DATA);
    parity++;
}
else
    UNSET(OPTICAL, OPTICAL_DATA);
waitUntilClockHigh();

// Parity Bit
waitUntilClockLow();
if ( parity & BIT0 )
    SET(OPTICAL, OPTICAL_DATA);
else
    UNSET(OPTICAL, OPTICAL_DATA);
waitUntilClockHigh();

// IDLE/stop bit
waitUntilClockLow();
UNSET(OPTICAL_DDR, OPTICAL_CLOCK);           // clock is input
UNSET(OPTICAL_DDR, OPTICAL_DATA);           // data is input

SET(OPTICAL, OPTICAL_DATA);                 // float data high
SET(OPTICAL, OPTICAL_CLOCK);               // float clock high
waitUntilClockHigh();

waitUntilClockLow();
waitUntilClockHigh();

// INHIBIT
SET(OPTICAL_DDR, OPTICAL_CLOCK);           // clock is output
UNSET(OPTICAL_DDR, OPTICAL_DATA);         // data is input

SET(OPTICAL, OPTICAL_DATA);                 // float data high
UNSET(OPTICAL, OPTICAL_CLOCK);           // pull clock low

```