Project

WindBot

Mikhail Bruk
Momchil Dimchev
Anton Talalayev

Mechatronics
ME5643

Introduction:

Wind turbines are designed to exploit wind energy at specific locations. Wind energy is plentiful, renewable, widely distributed and helps reduce greenhouse gas emissions. Turbine placement is usually preselected using a wind atlas and then validated using wind measurements on the field. In most cases, meteorological wind data alone is not sufficient for accurate placing of a large wind power project. Collection of terrain-specific data for wind speed and direction is crucial to determining a site's potential. There is no way to account for all the variables affecting the wind in the nature, so only experimentally collected data will be accurate enough. It is impractical to apply manpower to the long and repeating task of measuring wind values across a large area, so an automated approach should be developed. A WindBot will split the terrain into a grid of coordinate points and will measure data at each point, recording key points with highest wind speed.

Theory:

Our team has designed and implemented an alternative method to retrieving on-site data for wind speed measurements. The primary objective of the Basic Stamp 2 controlled autonomous robot is to travel around the field and read the wind speeds at various locations. In order to do that, the minimal design for the robot involved a propeller fitted with an encoder to record rotational speed. The top three wind speed measurements and their locations would be stored for later access. The measurements would be displayed on an LCD for a human operator to monitor. After surveying the site and tracking its own movements by the use of an accelerometer for better accuracy, the robot would return to the initial position. The human operator can then select any of the stored locations and direct the robot to go back for further testing. To keep track of the locations, robot would divide the area in grids, assign coordinates to them and measure the wind speed for each grid. Depending on dimensions,

more data points can be acquired at a cost of increased time. The optimal design for this robot would also involve: a manual mode in which the user would be able to control the robot with an IR remote control, wind direction measurement, obstacle avoidance system and emergency safety shutdown button.

In the final design, the priority was given to the wind direction over other optimal design features. Final implementation included wind speed measurement, wind direction measurement, compass navigation, distance measurement, course correction, and ROM data storage. Obstacle avoidance and return to point of highest wind speed were not implemented. The discrepancy between initial and final design will be discussed in Analysis section of the report.

Mathematical Background:

Certain BASIC STAMP limitations required the programmer to come up with alternative ways to carry on certain calculations. Basic stamp operates with integers only, and any division or factor multiplication has to be carefully analyzed.

In order to multiply a number by a decimal fraction, a ** operator has to be used, such that PBASIC operates with whole numbers, and then scales down the result to match the required answer. In order to get the decimal point to work correctly, a remainder has to be used to get all the data. The following line of cod is an excerpt from the programming part of the project:

```
Debug DEC rpm_1 ** 145, ".",DEC ((rpm_1 // 455) * 100 ) ** 145   ,CR
```

First, the variable is multiplied by the conversion factor to get the whole part of the number, then the decimal point is displayed, and after that the part of the number after the decimal point is calculated using the remainder.

When dealing with angles, it is important to know the relative position of one direction with respect to another. If angle one is bigger than angle two, angle one is to the right. It is a simple comparison, unless you have a singularity point between two directions. Singularity point occurs between 360 and 0 degrees. In order to check for this case, an angle had to be compared to its opposite, 360 – angle. If the opposite is smaller, you are dealing with this specific case and have to compute the position accordingly. Here, the relationship between two angles is flipped: if angle one is bigger than angle two, angle one is to the left.

Mechanical Designs:

Initial design of the project began with research of modern day anemometers and their advantages and disadvantages.

The first considered design was a windmill based mechanical anemometer. This type implies that the axis of rotation is vertical and the velocity measuring axis is horizontal. This design included a propeller combined with a vane on the horizontal axis, attached to the freely rotating vertical axis. The benefit of this kind of anemometers is that they show both, speed and direction of the wind.

Figure 1 Windmill Based Anemometer

The next design that was considered was a cup anemometer. It consisted of three or more hemispherical cups attached on horizontal arms that are in turn attached to a vertical shaft at equal angles to each other. The advantage to such a design is that it is very simple and easy to construct. It also gives a very small error (less that 3% up to 60mph) in the wind speed measurements. A major disadvantage, though, is that it does not give a wind direction.

Figure 2 Cup Anemometer

After evaluating both ideas, and the possible complications of each mechanical design, it was decided that the windmill based anemometer was the more useful devise for the specified objective. It would result in a more complicated design with a benefit of measuring wind direction.

A single propeller, two vane design was chosen. Parallel vanes made of balsa wood were spaced at the distance equal to the propeller diameter. This ensured undisturbed airflow to position the propeller parallel to the wind direction (single vane design implies vane to be directly behind the propeller, where turbulence occurs).

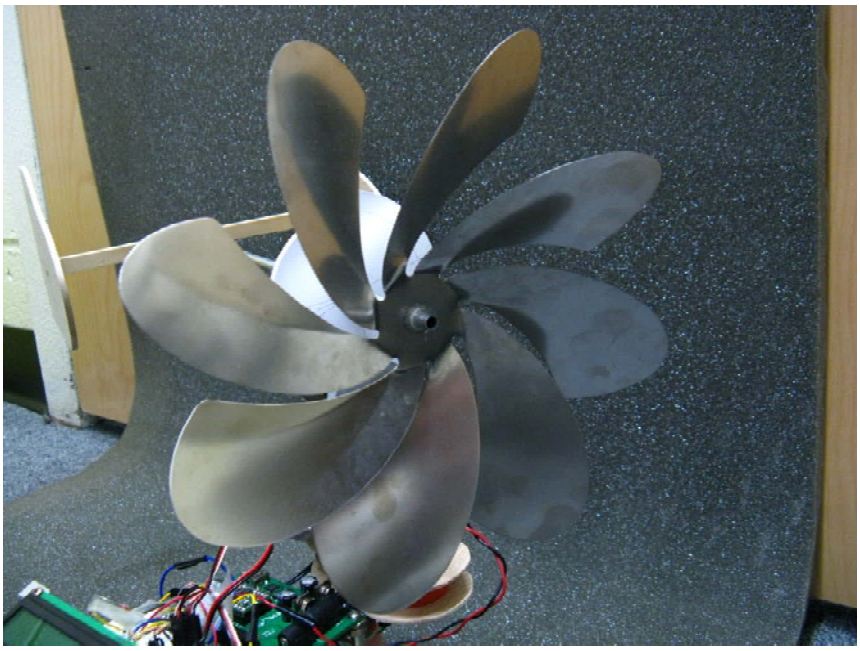

Figure 3 Propeller

To ensure free rotation of the anemometer, the structure had to be light. It was constructed out of inexpensive, readily available materials. After cost assessment, it was decided to purchase an aluminum propeller rather than manufacture one. The horizontal axis connecting the propeller and the vanes was made out of hollow aluminum rod. A cylindrical wooden rod was used for the vertical axis that joins the horizontal one and the moving base of the robot itself. This effective lightweight design

was tested under different wind conditions and met all design requirements. Wind was generated by a variety of fans ranging from small table fan to a large industrial floor fan.

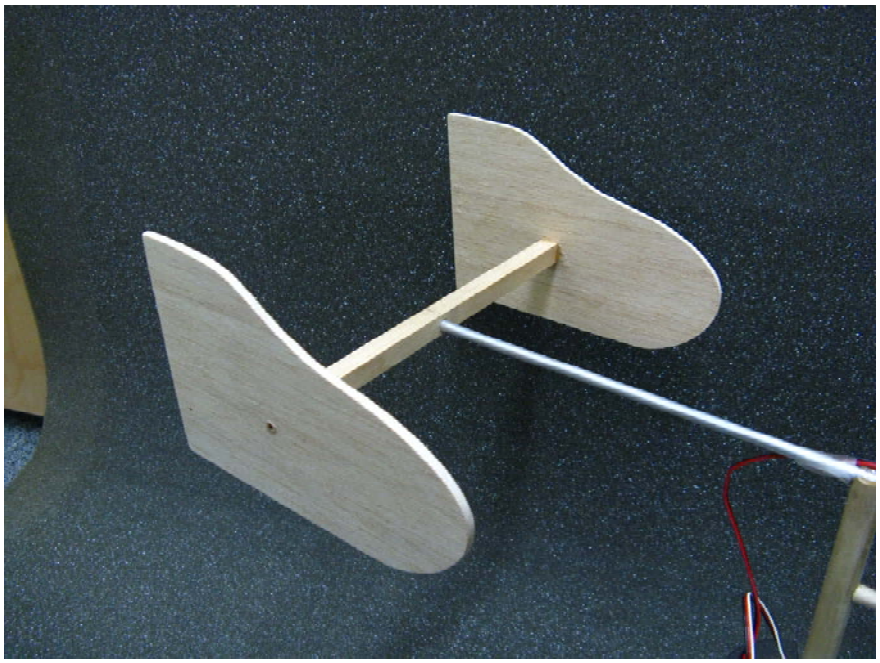

Figure 4 Vanes

Another important mechanical component was the chassis of the robot. It had to be sturdy enough to carry the previously described wind measuring device and big enough to support the Board of Education programming board, any additional bread boards, sensors and the two servo motors. Since a scale model was to be constructed, the team decided to use the high-quality brushed aluminum chassis from the Boe-Bot® robot kit.

Wind velocity is measured through and IR LED/Photoresistor tachometer. Initially, it was assumed that no light protection was necessary, but testing showed that tachometer behaves differently in different light environments. Since the WindBot is assumed to work under many conditions, the tachometer was placed in an enclosure canister.

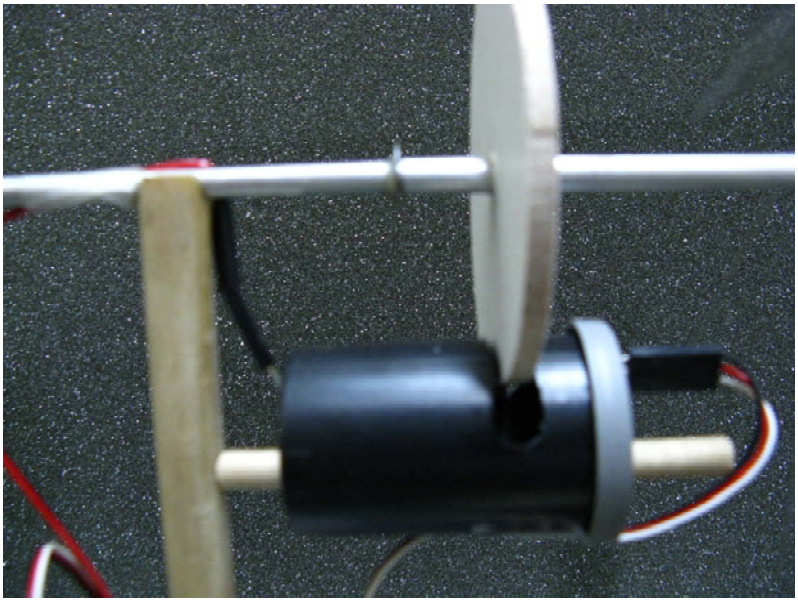

Figure 5 Tachometer

Due to final design decision, a sensor was required to calculate anemometer position with respect to the robot. An RC circuit was used with an LED/Photoresistor pair that measured RCTime through a thin film with 8 different transparency levels. The sensor was placed in an enclosure canister to ensure consistent data under all lighting conditions.

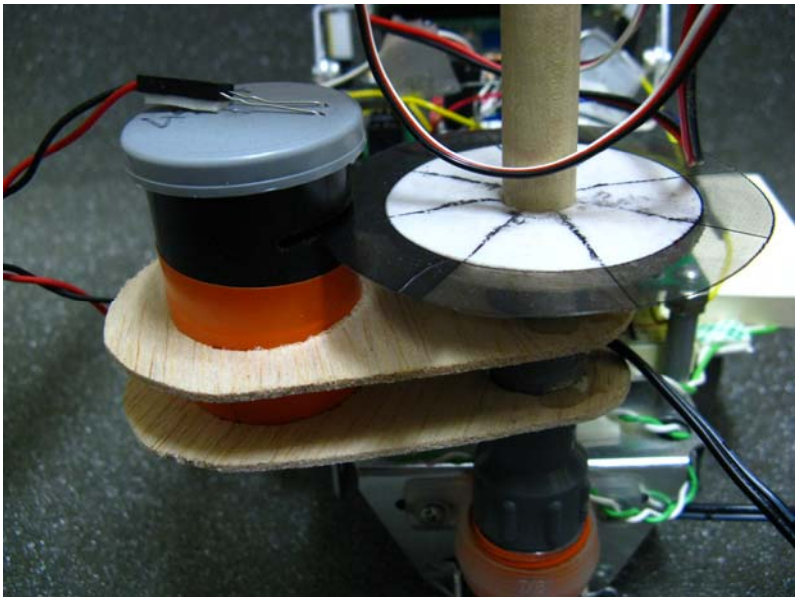

Figure 6 Enclosure canister of Wind Direction Sensor

Wheel encoders had to be constructed, so a set of Boe-Bot® kit wheels were used, since each wheel had 8 holes and was perfect for the task. That ensured a distance measurement equal to one eighth of a rotation. This design allowed for no slippage, so rubber tires were used to increase friction with the floor.

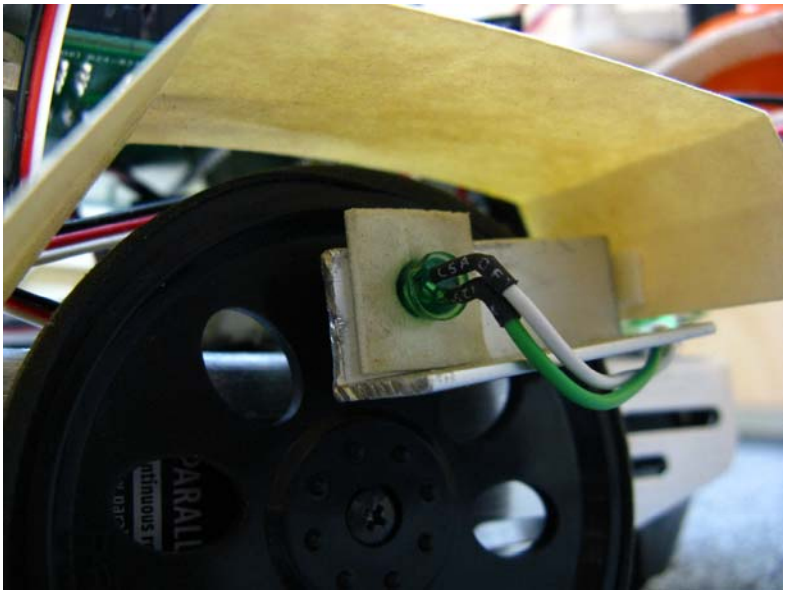

Figure 7 Wheel Encoder

Wheel encoders are affected by different light conditions, so an appropriate design solution had to be considered. Since most of the light comes from above, fenders were chosen as the best solution. It enables an excellent light protection and easy access to the sensor from the bottom for adjustments and repairs. Occam's razor principle was used, and simple, cardboard-made fenders were cut.

Obstacle avoidance would be implemented through the "whisker navigation" that involves two thin wires protruding from the front of the WindBot. After sufficient deflection, a circuit is closed, and a Basic Stamp PIN goes from low to high. Although, the mechanical sensor was constructed, this feature did not make it to the final robot design.

Electronic Circuits:

Wheel Rotation Sensors

In order for the robot to navigate the field and be able to keep track of the distance travelled in x and y directions, the use of rotary encoders was implemented on the wheels. Utilizing the wheel as an encoder disk that contains equidistant holes, an LED and photoresistor pair was used. The following electrical circuit was designed:

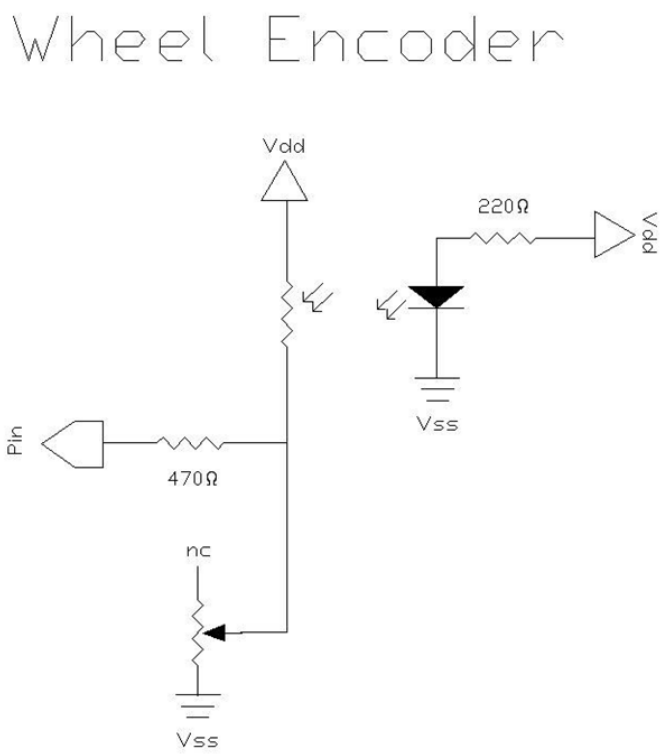

Figure 8 Electric Circuit

The LED is connected to Vdd through a 220 Ohm resistor, to ensure that the current that passes through the LED is below the maximum forward current (25mA). Facing the LED is a photoresistor that increases its conductivity every time the hole in the wheel is in front of it, allowing light to pass through. When the voltage at the PIN is below 1.4 V, it outputs LOW to the Basic Stamp. By connecting a potentiometer, the threshold value can be varied for different light conditions. By not connecting the second leg of the

potentiometer, it acts as a variable resistor and not a voltage divider. For safety, a 470 Ohm resistor is placed to protect the Basic Stamp from being short circuited in case the PIN is set HIGH accidentally.

Object Detection ("Whiskers Navigation")

The obstacle avoidance program relies on two "Whiskers" to detect whether or not there is an obstacle in front of the robot. The so called "whisker" acts as a Single Poll Single Throw Switch that closes and completes the circuit upon impact. The following circuit is used for each of the two "whiskers":

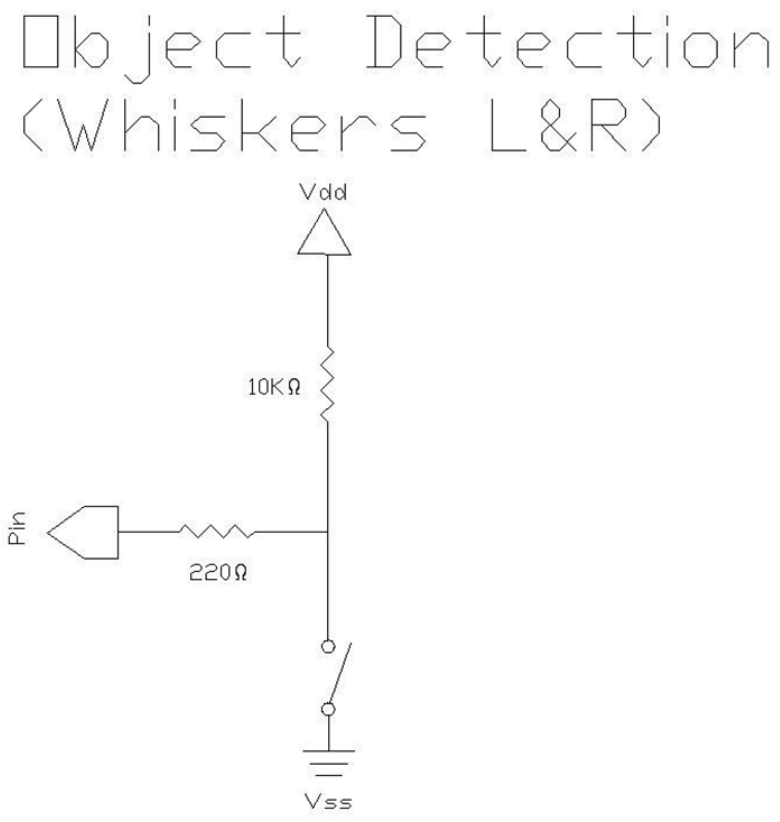

Figure 9 Electric Circuit

The PIN is set as an input pin and is being bulled HIGH by the 10 kOhm pull-up resistor when the switch is open. As soon as the switch closes, the pin is pulled LOW by Vss. For safety, a 220 Ohm resistor is placed to protect the Basic Stamp from being short circuited in case the PIN is set HIGH accidentally and the switch is closed.

Propeller Velocity (RPM)

For an accurate recording of the wind velocity a tachometer was constructed using an infrared emitter and receiver pair. The receiver is a transistor that acts as a photonic detector which reacts to the infrared radiation. The following circuit was constructed to perform the necessary operation.

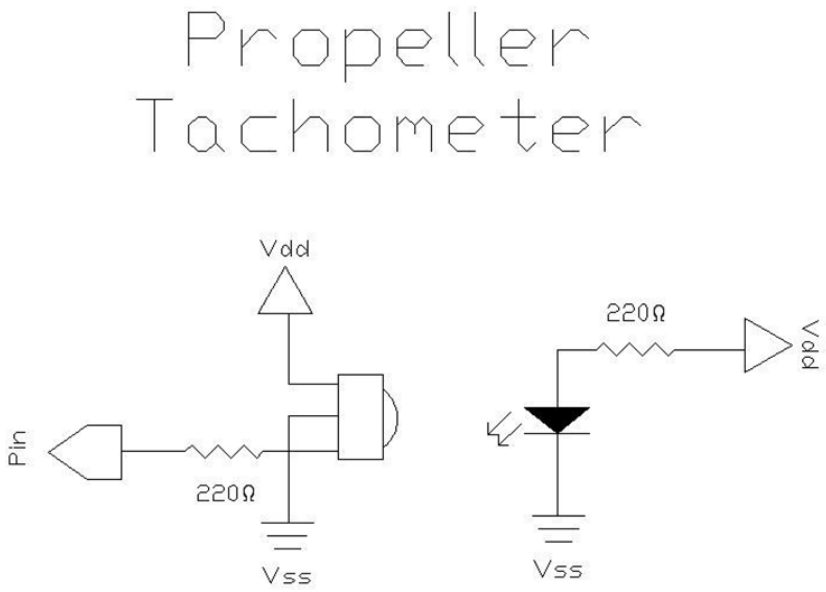

Figure 10 Electric Circuit

The IR emitter was positioned in front of the detector, with a divider placed between them. A 220 Ohm resistor was put in front of the infrared LED to ensure that the current does not burn it. The IR

detector has a built-in optical filter that allows very little light other than the 980 nm infrared that is being emitted. It also has an electric filter that only allows signal around 38.5 kHz to pass through. Every time the hole in the tachometer wheel is lined up with the detector and emitter, the detector outputs a HIGH to the PIN. Once again, for safety, a 220 Ohm resistor is placed to protect the Basic Stamp so that the current sunk into it is below the allowed 25 mA.

Wind Direction (RCTime)

For the robot to be an effective anemometer, it is also necessary to know the direction of the wind as well as its velocity. Therefore, an LED and photoresistor pair is utilized through the RCTime to determine direction. The following circuit was used:

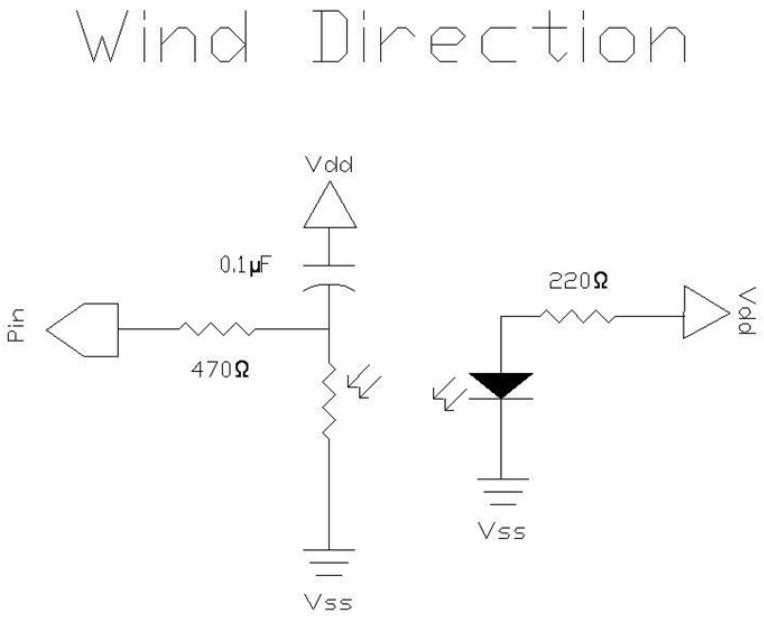

Figure 11 Electric Circuit

RCTime measures the time to charge the 0.1 µF capacitor through the resistor. In this case a photoresistor is used to vary the RCTime which depends on the transparency of the material between the LED and the Photoresistor. The higher the resistance, the longer the RCTime. The 470 Ohm resistor is used to protect the PIN in case it is accidentally made HIGH and shorted with ground.

Servo Motors' Control

The circuit for the Servo Motors involves a 555 timer IC microchip. In astable mode the 555 Timer generates a square pulse wave required by the servos to determine their speed. A properly sized capacitor and two resistor values were calculated to set the appropriate duty cycle. The following diagram represents the connections between the Servos, the 555 timer and the Basic Stamp:

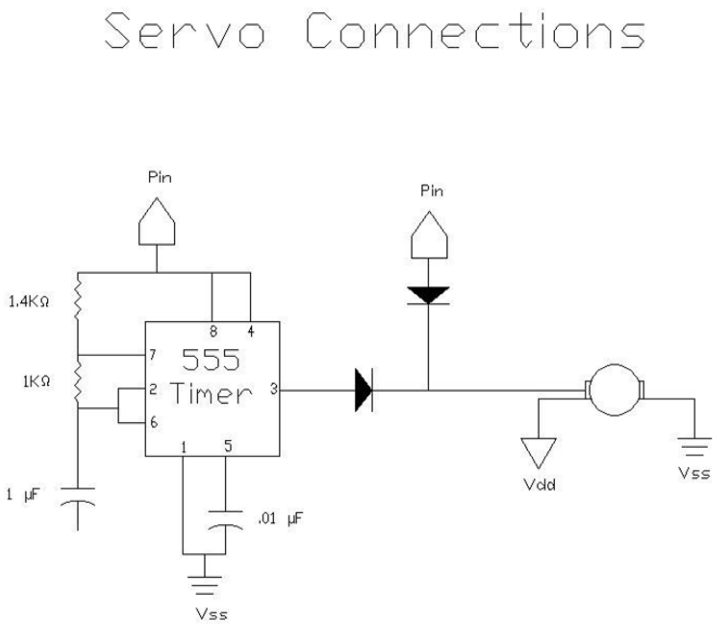

Figure 12 Electric Circuit

The time between two pulses from the 555 Timer is called t_{low} and the duration of each pulse is t_{high}.

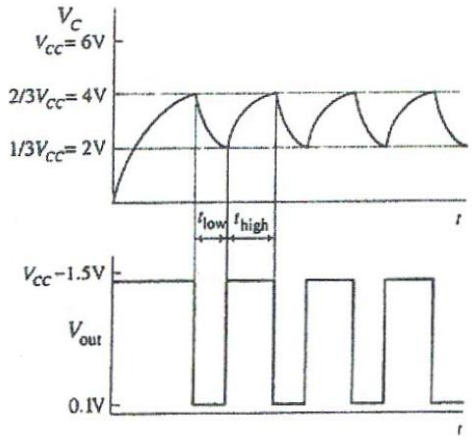

Figure 13 Pulse from 555 Timer

The t_{high} required by the servo motor to rotate forward is 1.7 ms and the time between pulses is 1ms. Next, the resistance values were determined in order for this pulse to be outputted. The following formulae were used in the calculation:

$$t_{low} = (0.693) * R_2 * C$$

$$t_{high} = (0.693) * C * (R_1 + R_2)$$

A 1 µF capacitor was used, combined with 1 ms for tl_{ow} and 1.7 ms for t_{high} and the following results were obtained:

$$R_2 = \frac{t_{low}}{(0.693) * C} \approx 1\ kOhm$$

$$R_1 = \frac{t_{high}}{(0.693) * C} - R_2 \approx 1.4\ kOhm$$

The 555 Timer is controlled by a pin and the servo motors can be controlled either by the 555 Timer or by a Pin. Normally in the design, the Servos receive a pulse from the 555 Timer; however, if it is required, the 555 Timer can be switched off and the Servos can be controlled by the PULSOUT command from a pin of the Basic Stamp. The two diodes are used to ensure that the circuit does not short.

Bill of Materials:

- **Boe-Bot Robot** $159.95
    - Chassis (1)
    - Wheels (2)
    - Servos (2)
    - Board of Education programming board (1)
    - BASIC Stamp® 2(1)
    - I/O Components**:**
        - LEDs (3)
        - Photoresistors (3)
        - Resistors (14)
        - Potentiometer (2)
        - Capacitors (4)
        - Infrared LEDs (1)
        - Receivers (1)
        - Whisker contact switches (2)
- **555 Timer IC** (2) $2.95

- **Power Supply**: (4 AA's) $2.99
- **Breadboard** (1) $7.99
- **LCD Terminal** $29.95
- **Hitachi HM55B Compass Module** (1) $29.95
- **Balsa Wood (1/8" x 6" x 36")** (1) $3.95
- **Tube Fitting** (1) $2.95
- **Aluminum Rod (1/16" x 12" x ¼")** (1) $2.36
- **Stainless Steel Propeller** (1) $10.95
- **Miscellaneous** $12.00
    - Solder
    - Glue
    - Electrical Tape
    - Double Sided Tape
    - Wires
    - Film Canisters (2)
    - Transparency (1)
    - Wooden Shaft
    - Button (1)

Total: $ 253.99

Prototype Cost:

In order to produce a functional WindBot prototype, an appropriate chassis has to be produced that will encase the meteorological equipment. Most of the cost will lie in the hull, as the sensors are relatively cheap. Increased computing power will allow for obstacle avoidance and multiple data point storage, with math software powering the analysis. A mounted video camera will allow a human operator to interfere in case of emergency and take over the controls.

The estimated cost of R&D, personnel, testing and materials is:

$ 5,000,000

Cost Analysis for Mass production:

The material and assembly cost is significantly lower than the prototype cost, and is estimated to be:

$ 50,000

Analysis:

The project design had four major components: mechanical, electrical, programming, and testing. Mechanical and electrical parts were done simultaneously, and once the robot was complete, the programming phase began.

A major drawback was developed, once the BASIC STAMP program had to do many things at the same time. A design change was suggested to change motor function from the PULSOUT command to the 555 timer, thus enabling an easier programming approach.

Ideally, the PULSOUT command sends a high pulse to the pin, followed by a PAUSE command that sends a low pulse. As the program gets more complicated and more code is introduced between PULSOUT commands, the motors begin to behave erratically because of the low pulse time change.

With the 555 timer, to turn the motor on, the PIN is set to high. When it has to stop, the PIN is set low. The disadvantages include a constant speed and inability to go backwards. Constant speed was not the issue, since the distance was measured based on the wheel rotation. To go backwards, it was decided to use an additional pin and PULSOUT command. Robot would go backwards only in the case of obstacle avoidance, which is assumed to happen rarely.

A certain variety of position sensing was considered. The most efficient solutions were a use of accelerometer, compass, or wheel encoders. Initially, accelerometer was picked as the best option, but extensive testing showed that it is accurate enough only in tilt measurements. Since most of the time robot moves at a constant speed, accelerometer would be ineffective. In order to calibrate accelerometer to the slight slope change of the floor, too much memory would be used.

Second solution was to use the wheel encoders, since they were already in the design for the distance measurement. After one of the wheels does a certain distance, the other would be checked, and any difference would be compensated by turning the robot. This method proved to be acceptable In following a straight line, but was difficult to implement when the WindBot had to turn 90 or 180 degrees. The encoder had a measurement error equal to the distance between two holes on the wheel, and the error propagated as the robot went further.

Final design solution used the digital compass to navigate through the field. It was as good as the encoders while going straight and excellent at turning. Due to the fact that real time data was gathered, the error was always the same, and within acceptable range. This would also prove very useful in obstacle avoidance, as the robot will always know its heading. The disadvantage was discovered

during testing, when it became clear that large metal objects such as steel tables offset the compass calibration, and throw the robot off course.

The main reason for deviation off course lies in the servomotors. They spin at the different rates, and require potentiometer tuning to reduce the error. By adding 555 timers, error was increased because resistor values were picked from readily available resistors, and were close to the desired values, but not exact. In order to solve this problem, a program was written, that measures the wheel rotation and displays the distance traveled by each wheel. By increasing time between the measurements, error is allowed to propagate to see better, which wheel is slower. Then it is manually corrected by twisting the potentiometer, until the distance values are close enough. (speed_alignment.bs2)

In order to measure wind speed in meters per second, a relationship had to be established between RPMs and m/s. To do this, a wind of known speed had to spin the blade. Using a fan provided too much error, as the air moved unevenly around the robot. A solution was developed, by placing anemometer on the RC Car. By driving it through a known distance and measuring time and average RPM, a plot was graphed that gave the conversion factor of 0.0022 from RPM to m/s. The y-intercept of 1.3 m/s gives the fan sensitivity threshold. (RC_car_test.bs2 and Retrieve_RC_car_test_data.bs2)

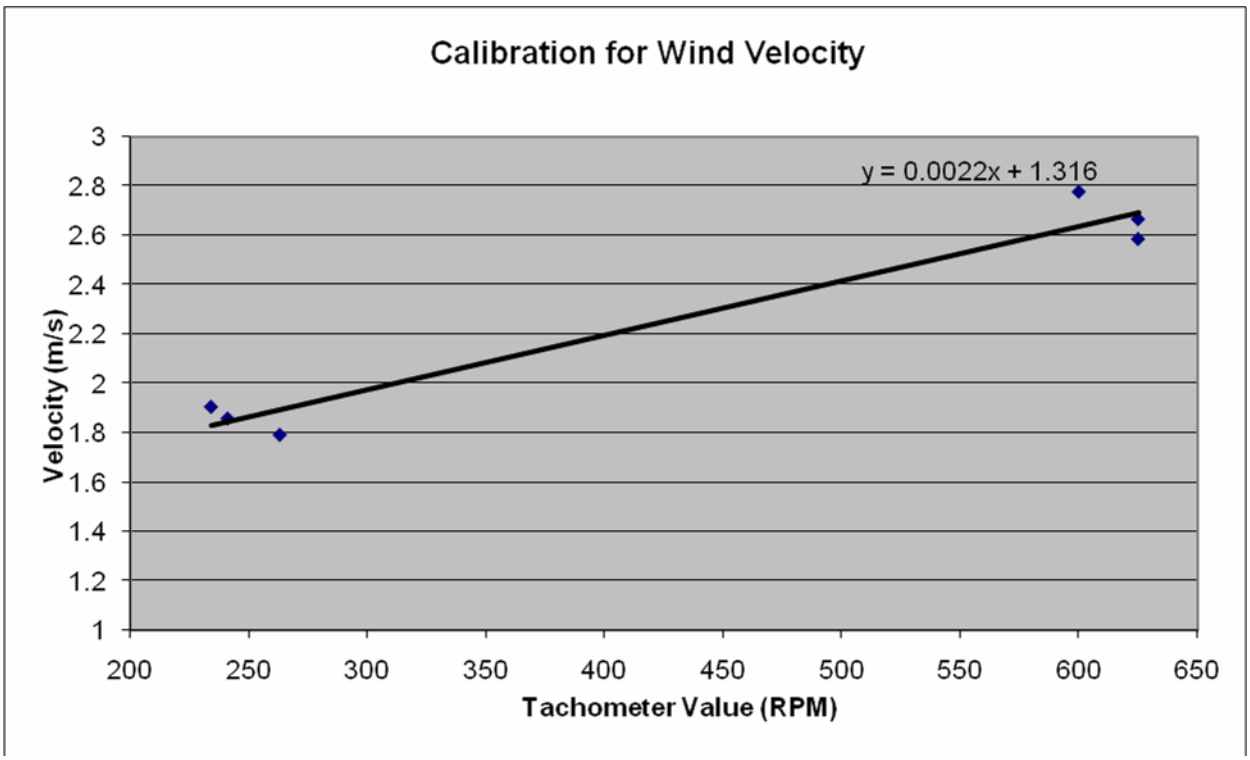

Figure 14 Plot for Calibration for Wind Velocity

In order to adjust the sensitivity of the encoders, a program was used that lets the user adjust the potentiometer, until he sees a steady data flow in every lighting environment. (wheel_encoder_adjustment.bs2)

Programming approach of the project was based on reusability and versatility. Since many of the robots functions were similar, subroutines were used to save valuable memory space, and allow for simper building blocks to be used while designing the robot path and action sequence. Versatility comes from the fact that program is structured to create the grid path with respect to any compass reading, and to change the grid size using one variable only.

In the beginning of the program, robot aligns the wheels equally, so the relative wheel to sensor position would be the same for both wheels. This reduces the error and creates a “cleaner” moving path. Then time is given to align the robot with the +y axis. Robot takes the compass reading, and uses it

as the bearing throughout the program. When robot turns 90 degrees left, it turns with respect to that bearing. This way the program is the same, whether you align it North or West, or any other way. As it measures the wind direction, it takes robot's direction into account, and calculates the wind direction with respect to earth's magnetic field.

The robot can follow a square grid with a side of any odd number. By using nested loops, a single variable controls the grid size. By using created subroutines, any shape and pattern can be easily created. The class demonstration uses a simplified pattern that was created in no time in PBASIC.

At this point, several more features had to be executed. Data had to be stored in the EEPROM and obstacle avoidance had to be implemented. Due to a lack of RAM memory, only two data points could be saved, with no obstacle avoidance.

The reason for a full RAM memory comes from the fact that a lot of Word variables had to be created for degree calculations. Digital compass used several variables to come up with an angle, and then it had to be compared with the robot bearing. Because of singularity point at North (360 degrees becomes 0 degrees), a more complicated code had to be produced. This is discussed in the Mathematical Background in greater detail.

In order to store the highest wind measurements, they would have to be compared with the old ones. Since RAM was full, some of the variables were reused. This is made program more confusing, and should be avoided whenever possible.

To retrieve the data, a second program is used, that reads EEPROM and displays the measurements on the debug screen.

The programs used are: "the_main_program.bs2" and "retrieve_wind_data.bs2".

The programs used for the class demonstration are: “class_demonstration.bs2” and “retrieve_wind_data.bs2”.

Throughout the project design and testing, a few valuable conclusions were made. Each sensor had to be tested many times under many conditions, because debugging the code required accurate data readings. All errors had to be accounted for and minimized whenever possible. A programming code that is aimed for a perfect robot would never work, as the robot would deviate from its course, require time to start receiving good date, or do something unpredictable due to environmental effects. Extensive testing had to be done to work everything out.

Appendix A:

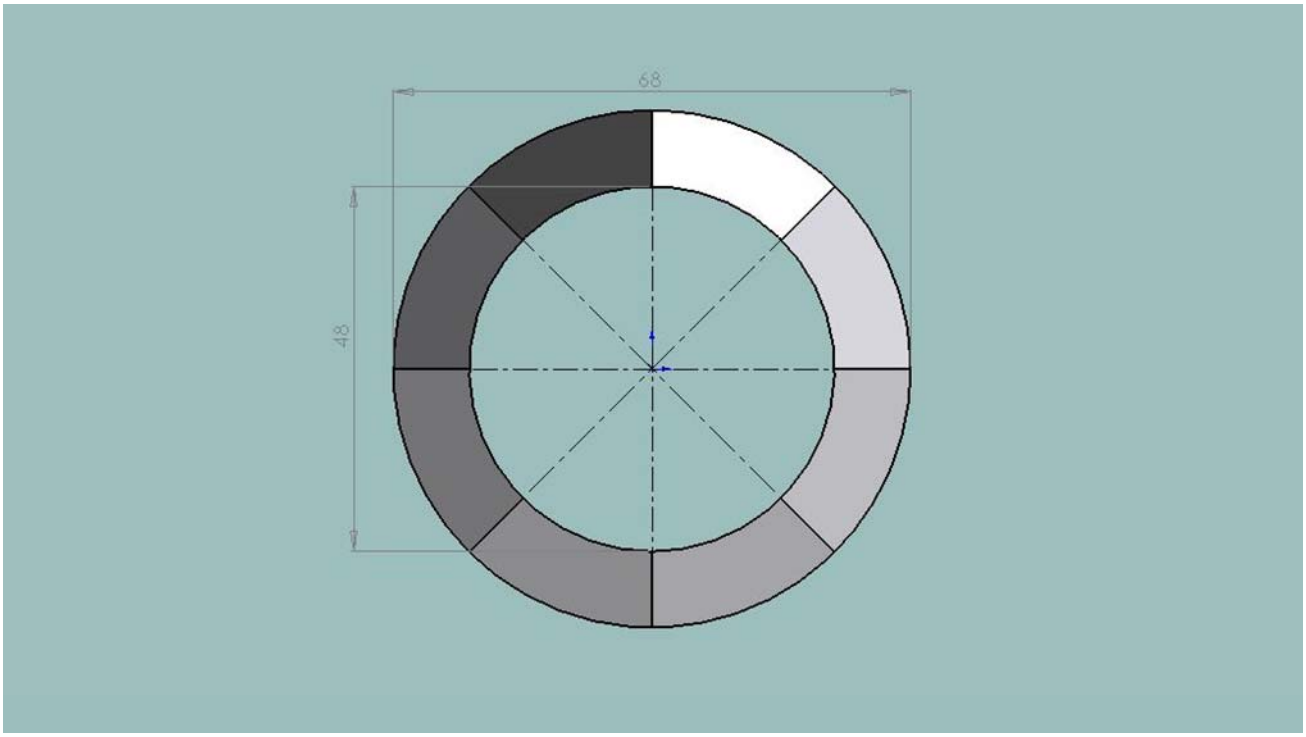

Figure 15 Encoder for Wind Direction

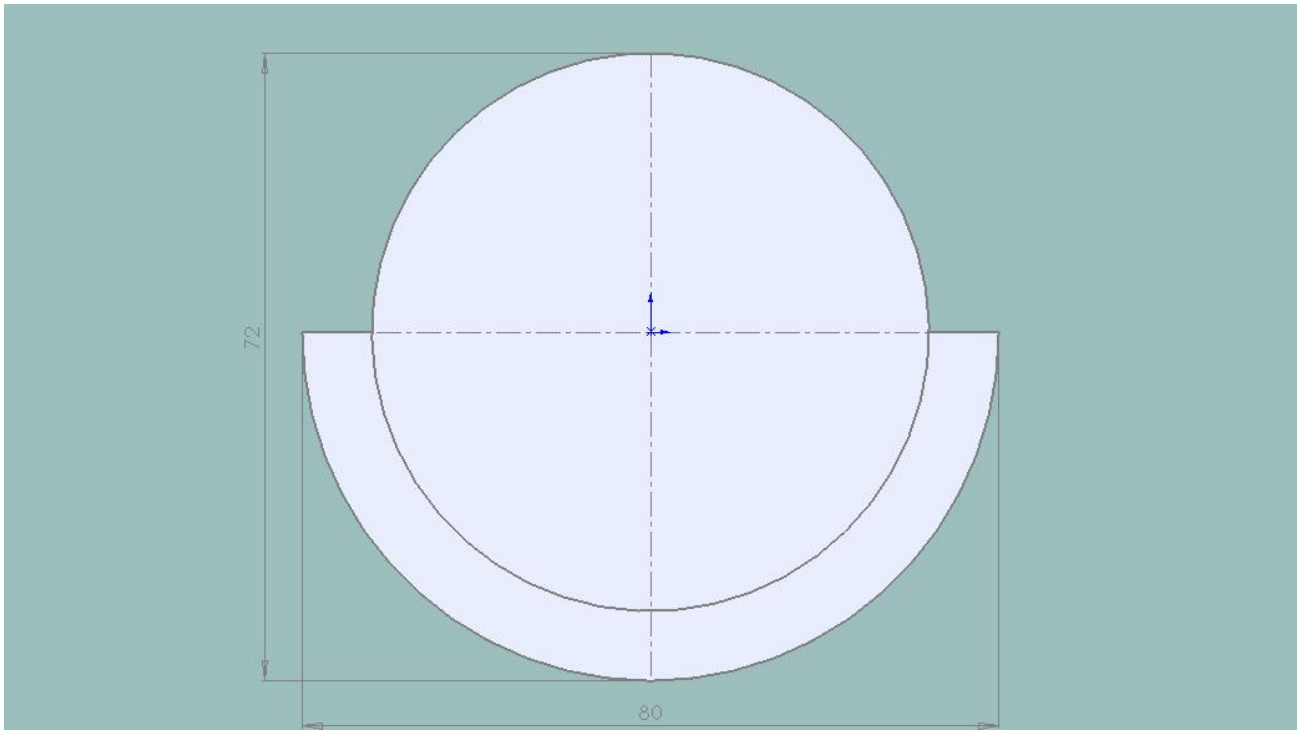

Figure 16 Encoder for Rotational Velocity

Appendix B:

Programs:

The_main_program.bs2:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' this program lets the robot go in a grid pattern, gather data, and store
the two best locations on EEPROM
' some variables are reused to save RAM space


' -----[ I/O Definitions ]-------------------------------------------------
-
DinDout PIN 8 ' P2 transceives to/from Din/Dout
Clk PIN 6 ' P0 sends pulses to HM55B's Clk
En PIN 7 ' P2 controls HM55B's /EN(ABLE)
' -----[ Constants ]-----------------------------------------------------
-
Reset CON %0000 ' Reset command for HM55B
Measure CON %1000 ' Start measurement command
```

```
Report CON %1100 ' Get status/axis values command
Ready CON %1100 ' 11 -> Done, 00 -> no errors
NegMask CON %1111100000000000 ' For 11-bit negative to 16-bits
' -----[ Variables ]-------------------------------------------------------
-
x VAR Word ' x-axis data
y VAR Word ' y-axis data
status VAR Nib ' Status flags
angle VAR Word ' Store angle measurement
bearing VAR Word 'Store angle direction
angle_one VAR Word     'course correction var and measure speed
angle_two VAR Word     'course correction var and EEPROM interaction
direction VAR Word     ' course correction var and EEPROM interaction
signal VAR Bit         'fan blade encoder var
prev_signal VAR Bit    'fan blade encoder var

right VAR Bit          'right wheel encoder
left VAR Bit           'left wheel encoder

right_wheel PIN 9      'powers right 555 timer
left_wheel PIN 10      'powers left 555 timer

counter VAR Nib        'alignment for loop counter and wind direction
measurememnt

counter_right VAR Nib  'right wheel travel distance
counter_left VAR Nib   'left wheel travel distance

prev_RPM VAR Word 'Wind measurememnt var
raw_revolutions VAR Word 'wind measurement var
angle_true VAR Word 'wind measurememnt

time VAR Word ' wind direction var


xloop VAR Nib 'square pattern var
yloop VAR Nib 'square pattern var
grid_size VAR Nib
grid_size = 4 ' actual grid size is grid_size + 1

counter_right = 0
counter_left = 0
raw_revolutions = 0
prev_signal = 0
prev_RPM = 0

DIR14 = 0
DIR2 = 0


' -----[ Initialization ]---------------------------------------------------
-
' LCD Initialization
PAUSE 200 ' Debounce power supply
SEROUT 13, 84, [22, 12] ' Turn on & clear LCD
PAUSE 5 ' 5 ms delay for clear cmd
```

```
'*********************WHEEL ALIGNMENT*************
'aligns the wheels for better initial accuracy
HIGH right_wheel
HIGH left_wheel


FOR counter = 0 TO 15
  right = IN2
  left = IN14
   PAUSE 20

  IF (right = 1 AND IN2 = 0) THEN
  LOW right_wheel
  ENDIF

  IF (left = 1 AND IN14 = 0) THEN
  LOW left_wheel
  ENDIF
NEXT


PAUSE 5000

'******************SET BEARING*******************
'sets the +y axis as the one robot is facing initially
'this lets the user to align the grid according to his wishes, not the
magnetic field direction
GOSUB Compass_Read
bearing = angle

'***********************ACTION*****************
'the program is assembled here

GOSUB Initialize_Data
GOSUB Square_Pattern

END

'---------------[ subroutine: Compass_Read ]------------------
'Calculates the angle the robot is facing
Compass_Read:
GOSUB Compass_Get_Axes ' Get x, and y values
angle = x ATN -y ' Convert x and y to brads
angle = angle */ 361 ' Convert brads to degrees
' LCD Display heading in degrees ON top line AND x AND y
 'measurements ON the bottom line.
SEROUT 13, 84, [128, " ", 128, DEC angle]
'151, " ", 151, SDEC X,
'160, " ", 160, SDEC y]
PAUSE 30 ' Debug delay for slower PCs
RETURN

' -----[ Subroutine - Compass_Get_Axes ]-----------------------------------
-
Compass_Get_Axes: ' Compass module subroutine
HIGH En: LOW En ' Send reset command to HM55B
SHIFTOUT DinDout,clk,MSBFIRST,[Reset\4]
```

```
HIGH En: LOW En ' HM55B start measurement command
SHIFTOUT DinDout,clk,MSBFIRST,[Measure\4]
status = 0 ' Clear previous status flags
DO ' Status flag checking loop
HIGH En: LOW En ' Measurement status command
SHIFTOUT DinDout,clk,MSBFIRST,[Report\4]
SHIFTIN DinDout,clk,MSBPOST,[Status\4] ' Get Status
LOOP UNTIL status = Ready ' Exit loop when status is ready
SHIFTIN DinDout,clk,MSBPOST,[x\11,y\11] ' Get x & y axis values
HIGH En ' Disable module
IF (y.BIT10 = 1) THEN y = y | NegMask ' Store 11-bits as signed word
IF (x.BIT10 = 1) THEN x = x | NegMask ' Repeat for other axis
RETURN

'------[ Subroutine - Correct_Course]--------------------------------------
'aligns the robot towards the bearing
Correct_Course:
DO
GOSUB Compass_Read
angle_one = ABS (bearing - angle)
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (bearing > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
ELSEIF (angle_one > angle_two) THEN
  IF (bearing > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN


'------[ Subroutine - Go_Straight]-----------------------------------------
'moves the robot forward one wheel rotation
Go_Straight:
counter_right = 0
counter_left = 0
HIGH right_wheel
HIGH left_wheel
DO                                              ' one rotation loop

  right = IN2
```

```
  left = IN14
   PAUSE 20

  IF (right = 1 AND IN2 = 0) THEN
  counter_right = counter_right + 1
  ENDIF


  IF (left = 1 AND IN14 = 0) THEN
  counter_left = counter_left + 1
  ENDIF

LOOP  UNTIL (counter_right > 5 OR counter_left > 5)   'rotation stops

LOW right_wheel
LOW left_wheel
RETURN

'------[ Subroutine: Measure_Wind_Speed ]------------------------
'gathers the RPM from the wind speed sensor
Measure_Wind_Speed:

PAUSE 3000
FOR angle_one = 0 TO 2000
  FREQOUT 15,1, 38500
  signal = IN15
'  DEBUG ? signal
  IF (signal = 1) AND (prev_signal = 0) THEN
    IF (raw_revolutions > 10) THEN
      DEBUG ? (15000/raw_revolutions)
      IF (prev_RPM < 15000/raw_revolutions) THEN
        prev_RPM = 15000/raw_revolutions
      ENDIF
    ENDIF
    raw_revolutions = 0
  ENDIF
  raw_revolutions = raw_revolutions + 1
  prev_signal = signal
NEXT
RETURN

'--------[ Subroutine: Square_Pattern ]----------------------
'makes the robot go in the grid
Square_Pattern:
GOSUB Go_Straight
GOSUB Correct_Course

FOR xloop = 0 TO grid_size

IF(xloop // 2 ) THEN
   FOR yloop = grid_size TO 0
    GOSUB Align_Back
    GOSUB Go_Straight
    GOSUB Measure_Wind_Speed
    GOSUB Wind_Direction
    GOSUB Store_Data
```

```
  NEXT
  GOSUB Align_Right
  GOSUB Go_Straight
  GOSUB Correct_Course
  GOSUB Measure_Wind_Speed
  GOSUB Wind_Direction
  GOSUB Store_Data

  ELSE
  FOR yloop = 0 TO grid_size
  GOSUB Correct_Course
  GOSUB Go_Straight
  GOSUB Measure_Wind_Speed
  GOSUB Wind_Direction
  GOSUB Store_Data

  NEXT
  GOSUB Align_Right
  GOSUB Go_Straight
  GOSUB Align_Back

  ENDIF
NEXT

  FOR xloop = grid_size + 2 TO 0
  GOSUB Align_Left
  GOSUB Go_Straight
  NEXT
  FOR yloop = (grid_size - 1) TO 0
  GOSUB Align_Back
  GOSUB Go_Straight
  NEXT

RETURN

'------------[ Subroutine: Align_Right ]-------------------------
'turns the robot bearing + 90
Align_Right:
DO
GOSUB Compass_Read
IF (bearing > 269) THEN
  direction = bearing - 270
ELSE
  direction = bearing + 90
ENDIF
angle_one = ABS (direction - angle)
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (direction > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
```

```
ELSEIF (angle_one > angle_two) THEN
  IF (direction  > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN

'------------[ Subroutine: Align_Left ]-------------------------
'turns the robot bearing - 90
Align_Left:
DO
GOSUB Compass_Read
IF (bearing < 91) THEN
  direction = bearing + 270
ELSE
  direction = bearing - 90
ENDIF
angle_one = ABS (direction - angle)
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (direction > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
ELSEIF (angle_one > angle_two) THEN
  IF (direction  > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN

'------------[ Subroutine: Align_Back ]-------------------------
'turns the robot bearing + 180
Align_Back:
DO
GOSUB Compass_Read
```

```
IF (bearing < 180) THEN
  direction = bearing + 180
ELSE
  direction = bearing - 180
ENDIF
angle_one = ABS (direction - angle)
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (direction > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
ELSEIF (angle_one > angle_two) THEN
  IF (direction  > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN

'-------------[ Subroutine: Wind_Direction ]----------------
'measures the aerovane direction with respect to earths magnetic field
Wind_Direction:
'DO
GOSUB Compass_Read
PAUSE 1000
  HIGH 0
  PAUSE 100
  RCTIME 0, 1, time
  DEBUG ? time
  IF (time >= 0 AND time < 340)  THEN
     DEBUG "1", CR
     counter = 0
  ELSEIF (time >= 340 AND time < 395 ) THEN
     DEBUG "2", CR
     counter = 1
  ELSEIF (time >= 395 AND time < 430) THEN
     DEBUG "3", CR
     counter = 2
  ELSEIF (time >= 430 AND time < 480) THEN
     DEBUG "4", CR
     counter = 3
  ELSEIF (time >= 480 AND time < 530) THEN
     DEBUG "5", CR
     counter = 4
```

```
  ELSEIF (time >= 530 AND time < 600) THEN
     DEBUG "6", CR
     counter = 5
  ELSEIF (time >= 600 AND time < 675) THEN
     DEBUG "7", CR
     counter = 6
  ELSE
     DEBUG "8", CR
     counter = 7
ENDIF
DEBUG ? time
DEBUG ? angle
DEBUG ? counter

IF (angle + (counter * 45)) < 360 THEN
angle_true =  (angle + (counter * 45))
ELSE
angle_true =  (angle + (counter * 45) - 360)
ENDIF
SEROUT 13, 84, [148, " ", 148, DEC angle_true, "   "]
PAUSE 1000
'LOOP
RETURN

'---------------------[ Subroutine: Store_Data]-------------------
'overwrites the data if its better than what is already stored
Store_Data:
READ 0, Word angle_one
READ 6, Word angle_two


IF (angle_one < angle_two) AND (prev_RPM > angle_one) THEN
  WRITE 0, Word prev_RPM, Word angle_true, xloop, yloop
ELSEIF (angle_two < angle_one) AND (prev_RPM > angle_two) THEN
  WRITE 6, Word prev_RPM, Word angle_true, xloop, yloop
ENDIF
prev_RPM = 0
RETURN

'----------------------[ Subroutine: Initialize_Data ]----------------------
-
'clears EEPROM
Initialize_Data:
DATA 0,0,0,0,0,0,0,0,0,0,0,0
RETURN
```

Class_demonstration.bs2:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' this program lets the robot go in a grid pattern, gather data, and store
the two best locations on EEPROM
' some variables are reused to save RAM space


' -----[ I/O Definitions ]--------------------------------------------------
-
DinDout PIN 8 ' P2 transceives to/from Din/Dout
Clk PIN 6 ' P0 sends pulses to HM55B's Clk
En PIN 7 ' P2 controls HM55B's /EN(ABLE)
' -----[ Constants ]-------------------------------------------------------
-
Reset CON %0000 ' Reset command for HM55B
Measure CON %1000 ' Start measurement command
Report CON %1100 ' Get status/axis values command
Ready CON %1100 ' 11 -> Done, 00 -> no errors
NegMask CON %1111100000000000 ' For 11-bit negative to 16-bits
' -----[ Variables ]-------------------------------------------------------
-
x VAR Word ' x-axis data
y VAR Word ' y-axis data
status VAR Nib ' Status flags
angle VAR Word ' Store angle measurement
bearing VAR Word 'Store angle direction
angle_one VAR Word     'course correction var and measure speed
angle_two VAR Word     'course correction var and EEPROM interaction
direction VAR Word     ' course correction var and EEPROM interaction
signal VAR Bit         'fan blade encoder var
prev_signal VAR Bit    'fan blade encoder var

right VAR Bit          'right wheel encoder
left VAR Bit           'left wheel encoder

right_wheel PIN 9      'powers right 555 timer
left_wheel PIN 10      'powers left 555 timer

counter VAR Nib        'alignment for loop counter and wind direction
measurememnt

counter_right VAR Nib  'right wheel travel distance
counter_left VAR Nib   'left wheel travel distance

prev_RPM VAR Word 'Wind measurememnt var
raw_revolutions VAR Word 'wind measurement var
angle_true VAR Word 'wind measurememnt

time VAR Word ' wind direction var
```

```
xloop VAR Nib 'square pattern var
yloop VAR Nib 'square pattern var
grid_size VAR Nib
grid_size = 4 ' actual grid size is grid_size + 1

counter_right = 0
counter_left = 0
raw_revolutions = 0
prev_signal = 0
prev_RPM = 0

DIR14 = 0
DIR2 = 0


' -----[ Initialization ]----------------------------------------------------
-
' LCD Initialization
PAUSE 200 ' Debounce power supply
SEROUT 13, 84, [22, 12] ' Turn on & clear LCD
PAUSE 5 ' 5 ms delay for clear cmd

'**********************WHEEL ALIGNMENT*************
'aligns the wheels for better initial accuracy
HIGH right_wheel
HIGH left_wheel


FOR counter = 0 TO 15
  right = IN2
  left = IN14
   PAUSE 20

  IF (right = 1 AND IN2 = 0) THEN
  LOW right_wheel
  ENDIF

  IF (left = 1 AND IN14 = 0) THEN
  LOW left_wheel
  ENDIF
NEXT


PAUSE 5000

'******************SET BEARING*******************
'sets the +y axis as the one robot is facing initially
'this lets the user to align the grid according to his wishes, not the
magnetic field direction
GOSUB Compass_Read
bearing = angle

'**********************ACTION*****************
'the program is assembled here

GOSUB Initialize_Data
```

```
GOSUB Go_Straight
GOSUB Correct_Course
GOSUB Go_Straight
GOSUB Correct_Course
GOSUB Go_Straight
GOSUB Correct_Course
xloop = 0
yloop = 0
GOSUB Measure_Wind_Speed
GOSUB Wind_Direction
GOSUB Store_Data

GOSUB Initialize_Data
GOSUB Go_Straight
GOSUB Correct_Course
GOSUB Go_Straight
GOSUB Correct_Course
GOSUB Go_Straight
GOSUB Correct_Course
xloop = 0
yloop = 1
GOSUB Measure_Wind_Speed
GOSUB Wind_Direction
GOSUB Store_Data

GOSUB Align_Right
GOSUB Go_Straight
GOSUB Align_Right
GOSUB Go_Straight
GOSUB Align_Right
GOSUB Go_Straight
GOSUB Align_Back

xloop = 1
yloop = 1
GOSUB Measure_Wind_Speed
GOSUB Wind_Direction
GOSUB Store_Data

GOSUB Go_Straight
GOSUB Align_Back
GOSUB Go_Straight
GOSUB Align_Back
GOSUB Go_Straight
GOSUB Align_Back

xloop = 1
yloop = 0
GOSUB Measure_Wind_Speed
GOSUB Wind_Direction
GOSUB Store_Data

GOSUB Go_Straight
GOSUB Align_Back
GOSUB Go_Straight
GOSUB Align_Back
GOSUB Go_Straight
```

```
GOSUB Align_Back

GOSUB Align_Left
GOSUB Go_Straight
GOSUB Align_Left
GOSUB Go_Straight
GOSUB Align_Left
GOSUB Go_Straight


END

'---------------[ subroutine: Compass_Read ]------------------
'Calculates the angle the robot is facing
Compass_Read:
GOSUB Compass_Get_Axes ' Get x, and y values
angle = x ATN -y ' Convert x and y to brads
angle = angle */ 361 ' Convert brads to degrees
' LCD Display heading in degrees ON top line AND x AND y
 'measurements ON the bottom line.
SEROUT 13, 84, [128, " ", 128, DEC angle]
'151, " ", 151, SDEC X,
'160, " ", 160, SDEC y]
PAUSE 30 ' Debug delay for slower PCs
RETURN

' -----[ Subroutine - Compass_Get_Axes ]-------------------------------------
-
Compass_Get_Axes: ' Compass module subroutine
HIGH En: LOW En ' Send reset command to HM55B
SHIFTOUT DinDout,clk,MSBFIRST,[Reset\4]
HIGH En: LOW En ' HM55B start measurement command
SHIFTOUT DinDout,clk,MSBFIRST,[Measure\4]
status = 0 ' Clear previous status flags
DO ' Status flag checking loop
HIGH En: LOW En ' Measurement status command
SHIFTOUT DinDout,clk,MSBFIRST,[Report\4]
SHIFTIN DinDout,clk,MSBPOST,[Status\4] ' Get Status
LOOP UNTIL status = Ready ' Exit loop when status is ready
SHIFTIN DinDout,clk,MSBPOST,[x\11,y\11] ' Get x & y axis values
HIGH En ' Disable module
IF (y.BIT10 = 1) THEN y = y | NegMask ' Store 11-bits as signed word
IF (x.BIT10 = 1) THEN x = x | NegMask ' Repeat for other axis
RETURN

'------[ Subroutine - Correct_Course]----------------------------------------
'aligns the robot towards the bearing
Correct_Course:
DO
GOSUB Compass_Read
angle_one = ABS (bearing - angle)
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (bearing > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
```

```
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
ELSEIF (angle_one > angle_two) THEN
  IF (bearing > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN


'------[ Subroutine - Go_Straight]-------------------------------------
'moves the robot forward one wheel rotation
Go_Straight:
counter_right = 0
counter_left = 0
HIGH right_wheel
HIGH left_wheel
DO                                          ' one rotation loop

  right = IN2
  left = IN14
   PAUSE 20

  IF (right = 1 AND IN2 = 0) THEN
  counter_right = counter_right + 1
  ENDIF


  IF (left = 1 AND IN14 = 0) THEN
  counter_left = counter_left + 1
  ENDIF

LOOP  UNTIL (counter_right > 5 OR counter_left > 5)   'rotation stops

LOW right_wheel
LOW left_wheel
RETURN

'------[ Subroutine: Measure_Wind_Speed ]------------------------
'gathers the RPM from the wind speed sensor
Measure_Wind_Speed:

PAUSE 3000
FOR angle_one = 0 TO 2000
  FREQOUT 15,1, 38500
  signal = IN15
```

```
'  DEBUG ? signal
  IF (signal = 1) AND (prev_signal = 0) THEN
    IF (raw_revolutions > 10) THEN
      DEBUG ? (15000/raw_revolutions)
      IF (prev_RPM < 15000/raw_revolutions) THEN
        prev_RPM = 15000/raw_revolutions
      ENDIF
    ENDIF
    raw_revolutions = 0
  ENDIF
  raw_revolutions = raw_revolutions + 1
  prev_signal = signal
NEXT
RETURN

'--------[ Subroutine: Square_Pattern ]----------------------
'makes the robot go in the grid
Square_Pattern:
GOSUB Go_Straight
GOSUB Correct_Course

FOR xloop = 0 TO grid_size

IF(xloop // 2 ) THEN
   FOR yloop = grid_size TO 0
    GOSUB Align_Back
    GOSUB Go_Straight
    GOSUB Measure_Wind_Speed
    GOSUB Wind_Direction
    GOSUB Store_Data

  NEXT
  GOSUB Align_Right
  GOSUB Go_Straight
  GOSUB Correct_Course
  GOSUB Measure_Wind_Speed
  GOSUB Wind_Direction
  GOSUB Store_Data

  ELSE
  FOR yloop = 0 TO grid_size
  GOSUB Correct_Course
  GOSUB Go_Straight
  GOSUB Measure_Wind_Speed
  GOSUB Wind_Direction
  GOSUB Store_Data

  NEXT
  GOSUB Align_Right
  GOSUB Go_Straight
  GOSUB Align_Back

  ENDIF
NEXT

  FOR xloop = grid_size + 2 TO 0
  GOSUB Align_Left
```

```
  GOSUB Go_Straight
  NEXT
  FOR yloop = (grid_size - 1) TO 0
  GOSUB Align_Back
  GOSUB Go_Straight
  NEXT

RETURN

'------------[ Subroutine: Align_Right ]-------------------------
'turns the robot bearing + 90
Align_Right:
DO
GOSUB Compass_Read
IF (bearing > 269) THEN
  direction = bearing - 270
ELSE
  direction = bearing + 90
ENDIF
angle_one = ABS (direction - angle)
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (direction > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
ELSEIF (angle_one > angle_two) THEN
  IF (direction  > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN

'------------[ Subroutine: Align_Left ]-------------------------
'turns the robot bearing - 90
Align_Left:
DO
GOSUB Compass_Read
IF (bearing < 91) THEN
  direction = bearing + 270
ELSE
  direction = bearing - 90
ENDIF
angle_one = ABS (direction - angle)
```

```
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (direction > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
ELSEIF (angle_one > angle_two) THEN
  IF (direction  > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN

'------------[ Subroutine: Align_Back ]-------------------------
'turns the robot bearing + 180
Align_Back:
DO
GOSUB Compass_Read
IF (bearing < 180) THEN
  direction = bearing + 180
ELSE
  direction = bearing - 180
ENDIF
angle_one = ABS (direction - angle)
angle_two = ABS (360 - angle_one)
IF (angle_one < angle_two)  THEN
  IF (direction > angle) THEN
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
  ELSE
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ENDIF
ELSEIF (angle_one > angle_two) THEN
  IF (direction  > angle) THEN
  HIGH left_wheel
  PAUSE 10
  LOW left_wheel
  ELSE
  HIGH right_wheel
  PAUSE 10
  LOW right_wheel
```

```
  ENDIF
ENDIF

LOOP UNTIL (angle_one < 5 OR angle_two < 5)
RETURN

'-------------[ Subroutine: Wind_Direction ]----------------
'measures the aerovane direction with respect to earths magnetic field
Wind_Direction:
'DO
GOSUB Compass_Read
PAUSE 1000
  HIGH 0
  PAUSE 100
  RCTIME 0, 1, time
  DEBUG ? time
  IF (time >= 0 AND time < 340)  THEN
     DEBUG "1", CR
     counter = 0
  ELSEIF (time >= 340 AND time < 395 ) THEN
     DEBUG "2", CR
     counter = 1
  ELSEIF (time >= 395 AND time < 430) THEN
     DEBUG "3", CR
     counter = 2
  ELSEIF (time >= 430 AND time < 480) THEN
     DEBUG "4", CR
     counter = 3
  ELSEIF (time >= 480 AND time < 530) THEN
     DEBUG "5", CR
     counter = 4
  ELSEIF (time >= 530 AND time < 600) THEN
     DEBUG "6", CR
     counter = 5
  ELSEIF (time >= 600 AND time < 675) THEN
     DEBUG "7", CR
     counter = 6
  ELSE
     DEBUG "8", CR
     counter = 7
ENDIF
DEBUG ? time
DEBUG ? angle
DEBUG ? counter

IF (angle + (counter * 45)) < 360 THEN
angle_true =  (angle + (counter * 45))
ELSE
angle_true =  (angle + (counter * 45) - 360)
ENDIF
SEROUT 13, 84, [148, " ", 148, DEC angle_true, "   "]
PAUSE 1000
'LOOP
RETURN

'--------------------[ Subroutine: Store_Data]-------------------
'overwrites the data if its better than what is already stored
```

```
Store_Data:
READ 0, Word angle_one
READ 6, Word angle_two


IF (angle_one < angle_two) AND (prev_RPM > angle_one) THEN
  WRITE 0, Word prev_RPM, Word angle_true, xloop, yloop
ELSEIF (angle_two < angle_one) AND (prev_RPM > angle_two) THEN
  WRITE 6, Word prev_RPM, Word angle_true, xloop, yloop
ENDIF
prev_RPM = 0
RETURN

'-----------------------[ Subroutine: Initialize_Data ]----------------------
-
'clears EEPROM
Initialize_Data:
DATA 0,0,0,0,0,0,0,0,0,0,0,0
RETURN
```

Retrieve_wind_data.bs2:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'this program retrieves the field data and displays the values

rpm_1 VAR Word
rpm_2 VAR Word

direction_1 VAR Word
direction_2 VAR Word



x_1 VAR Nib
x_2 VAR Nib

y_1 VAR Nib
y_2 VAR Nib



READ 0 ,Word rpm_1, Word direction_1, x_1, y_1, Word rpm_2, Word direction_2,
x_2, y_2
PAUSE 500
DEBUG "**************WIND MEASUREMENT DATA:**************",CR,
      "DATA POINT ONE:",CR,
      "COORDINATES:    (", DEC y_1, ",", DEC x_1, ")",CR,
      "WIND VELOCITY:  ",DEC rpm_1 ** 145, ".",DEC ((rpm_1 // 455) * 100 ) **
145   ,CR,
      "WIND DIRECTION: ", DEC direction_1, CR,CR,
      "DATA POINT TWO:",CR,CR,
      "COORDINATES:    (", DEC y_2, ",", DEC x_2, ")",CR,
      "WIND VELOCITY:  ", DEC rpm_2 ** 145, ".",DEC ((rpm_2 // 455) * 100 )
** 145   ,CR,
      "WIND DIRECTION: ", DEC direction_2, CR

END
```

Rc_car_test.bs2:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' this program measures the wind speed while on the RC car, and records the
highest value


signal VAR Bit
prev_signal VAR Bit
counter VAR Word
prev_counter VAR Word

counter = 0
prev_signal = 0
prev_counter = 0

PAUSE 5000

HIGH 2

DO
  FREQOUT 15,1, 38500
  signal = IN15
'  DEBUG ? signal
  IF (signal = 1) AND (prev_signal = 0) THEN
    IF (counter > 10) THEN                              'counter increases
while signal is not changing
      DEBUG ? (15000/counter)                           'once signal changes,
RPM is calculated and counter is reset
      IF (prev_counter < 15000/counter) THEN
        prev_counter = 15000/counter                    ' 15000 is the
conversion factor for the RPM
      ENDIF
      DEBUG ? prev_counter
      WRITE 0, Word prev_counter
    ENDIF
    counter = 0
  ENDIF
  counter = counter + 1
  prev_signal = signal
LOOP
```

Retrieve_rc_car_test_data.bs2:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'this small program reads the data from locations 0 and 1
'and uses it to find the highest RPM during the RC car test


value VAR Word
READ 0, Word value

DEBUG ? value

END
```

Wheel_encoder_adjustment.bs2:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'this program lets the user adjust the potentiometer to appropriate lighting
conditions

DIR14 = 0
DIR2 = 0
value VAR Bit
counter_right VAR Word
counter_right = 0

HIGH 9
HIGH 10
DO
  value = IN2
'  DEBUG ? value
'  DEBUG ? IN2
PAUSE 14
  IF (value = 1 AND IN2 = 0) THEN
  counter_right = counter_right + 1
  ENDIF

LOOP UNTIL (counter_right > 7)
DEBUG ? counter_right
LOW 9
LOW 10
```

Speed_alignment.bs2:

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' this program lets the user to adjust the potentiometers on the servos to
amke wheels rotate at the same speed
' the wheel encoders show the rotations of both wheels, and as the values
approach each other, the motors spin at
' closer speeds

'***************INITIALIZATION******************
right VAR Bit 'stores the previous measurement
left VAR Bit  'stores the previous measurement


counter VAR Nib

counter_right VAR Word  'keeps the amount of revolutions
counter_left VAR Word   'keeps the amount of revolutions

counter_right = 0
counter_left = 0

DIR14 = 0
DIR2 = 0

'*********************WHEEL ALIGNMENT*************
HIGH 9
HIGH 10


FOR counter = 0 TO 15
  right = IN2
  left = IN14
   PAUSE 20

  IF (right = 1 AND IN2 = 0) THEN
  LOW 9
  ENDIF

  IF (left = 1 AND IN14 = 0) THEN
  LOW 10
  ENDIF
NEXT


PAUSE 1000


'***********************MOVEMENT*****************

HIGH 9
```

```
HIGH 10

DO
DO


  right = IN2
  left = IN14
   PAUSE 20

  IF (right = 1 AND IN2 = 0) THEN
  counter_right = counter_right + 1
  ENDIF


  IF (left = 1 AND IN14 = 0) THEN
  counter_left = counter_left + 1
  ENDIF







LOOP  UNTIL (counter_right > 20 OR counter_left > 20) 'when one of the wheels
do 20/8 rotations, program displays rotation of both wheels
  DEBUG CLS
  DEBUG ? counter_right
  DEBUG ? counter_left
  counter_left = 0
  counter_right = 0


LOOP
```

Bibliography:

- Lindsay, A. (2003-2004). *What's a Microcontroller?*, Parallax, Inc.
- Lindsay, A. (2003-2004). *Robotics with the Boe-Bot*, Parallax, Inc.
- Lindsay, A. (2003-2004). *Basic Analog and Digital*, Parallax, Inc.
- Lindsay, A. (2003-2004). *Smart Sensors & Applications*, Parallax, Inc.
- Kapila, V. (2008). *Mechatronics Lecture Slides*, Polytechnic University
- Parallax, Inc. (2007), http://www.parallax.com/