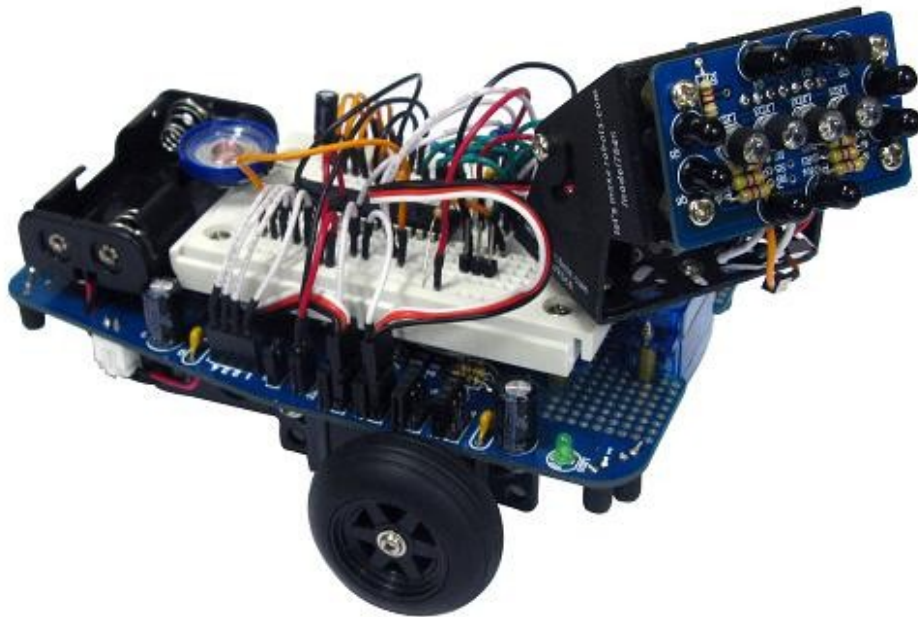


Understanding the Mr. General robot kit

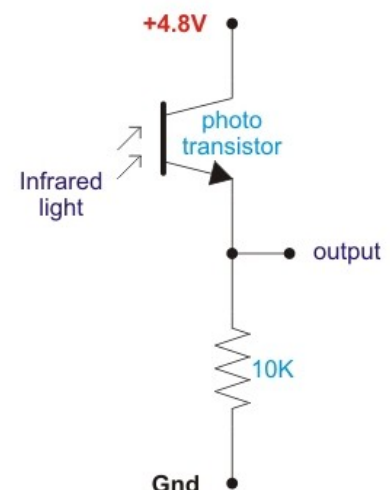


The sensors:

The sensors are all IR (infrared) sensors. IR sensors can receive interference from sunlight and indoor lighting. The Mr. General robot overcomes this problem by taking two readings, One with the IR LEDs on and another with the IR LEDs off. The difference is measured to determine IR reflected by an object. This will be explored later when we look at the code.

When you look at the corner sensors on Mr. General they are just a simple voltage divider. The principle is that the phototransistor works much like a LDR (light dependant resistor). As more light shines on the transistor it allows more current to flow. The voltage across the fixed resistor changes with the current flowing.

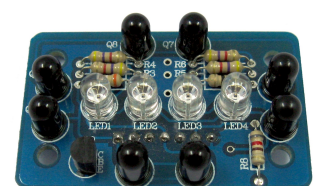
When the sensor is connected to a digital input then it will read as a 1 when the voltage goes above 60% of the CPU voltage and will read as a 0 when it falls below 40%. The actual voltage levels may vary slightly depending on the processor used. When connected to an analog input then the voltage will get higher as the object gets closer.



The advantage of using the sensor on an analog input is that your code can then have more than one level. You might have:

- 0 - 200 nothing there
- 201 - 500 something there but no chance of collision. Curious? have a look.
- 501 - 800 try to avoid this
- 801 - 1023 better back up or something quick!

The Eye: consist of 4 sensors almost the same as those on the corners of the main PCB. The main difference being that there are two phototransistors in parallel. This doubles the sensitivity. Increasing the brightness of the IR LED could also increase the sensitivity but uses more power and flattens the batteries quicker.



Servos - how they work:

Mr. General uses two standard micro servos for the pan/tilt assembly that makes his head/neck. This [pan/tilt kit was designed on LMR](#) (letsmakerobots.com) These servos consist of a small motor, a gearbox and a variable resistor connected to the output shaft. A small built in control circuit uses the variable resistor to determine what position the output shaft is in. A control pulse tells the circuit where the shaft should be. The control circuit uses this information to adjust the speed and direction of the servo motor until the position of the output shaft matches the position given by the control pulse.

This pulse is between 1mS and 2mS in width with 1.5mS being the center position of the servo. The pulse repeats every 20mS. Different servo manufacturers have slightly different standards and as such some servos will work with pulse widths outside of the 1-2mS range. Pulses that exceed these limits may force your servo to try and move beyond it's range of travel and possibly damage the servo.

Continuous rotation servos:

Mr. General uses two continuous rotation servos for movement. A continuous rotation servo is similar to a standard servo except that the output shaft can rotate continuously. For this reason there is no variable resistor that monitors the position of the output shaft like there is in a standard servo. The variable resistor is replaced with two fixed resistors that simulate the variable resistor set to the center position.

Now a pulse of 1.5mS will make the motor stop. A pulse greater than 1.5mS will make the output shaft turn clockwise and a pulse less than 1.5mS will rotate the shaft counter clockwise. The closer the pulse width is to 1.5mS the slower the motor will turn. In reality no two servos are the same and as such 1.5mS is only the theoretical center position. Temperature can also affect this center position causing the motors to drift slightly in one direction on a cold morning and the other direction on a hot day.

To connect either type of servo to Mr. General simply plug it into the 3 pin male headers provided with the ground wire (usually black or brown) to the outer edge of the PCB. The 3 pin male headers already have power supplied (4.8V) so only a single wire is needed to connect the signal wire to the processor. All 3 pin male headers have a 3 pin female header connected in parallel so that a jumper wire can be used to connect the servo to the breadboard.

Processor:

The Mr. General robot was designed to work with any processor that can plug into it's breadboard. For this document we will look at the Atmega8A using the Arduino bootloader and IDE as well as the Picaxe 28X1.

Making a programming cable for the ATmega8A:

To program the ATmega8A while it is in the breadboard will require a homemade programming cable. There are many USB to serial interfaces available. The photo below shows a typical example of what you might find in an office supply store. Alternatively you could buy a USB/serial breakout board such as the [FTDI basic from Sparkfun](#).



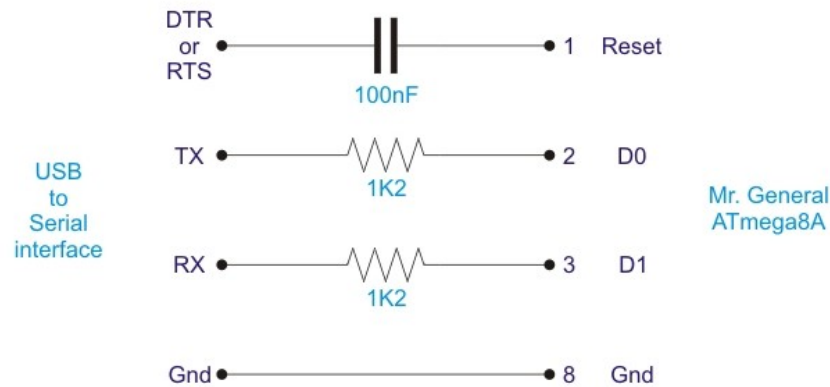
USB to Serial adaptor

Pin 5: Gnd
Pin 4: DTR
Pin 3: TXD
Pin 2: RXD

No matter which interface you use your cable will need 4 wires. Gnd, TX, RX and DTR or RTS. Normally I use DTR but if that doesn't work then try RTS. A full list of pinouts for the D9 serial socket can be found here: http://en.wikipedia.org/wiki/Serial_port

In order for the Arduino IDE to reset the ATmega8A prior to programming a 100nF capacitor needs to be in series with the DTR and reset pins. This capacitor allows the DTR pin to briefly trigger a reset allowing the bootloader to accept a new program.

I have shown 1K2 resistors in series with TX and RX. This is a simple precaution as D0 and D1 can be input or output depending on your program. D0 should be disconnected from the eye and D1 disconnected from the servo prior to program upload to ensure a reliable data transfer.



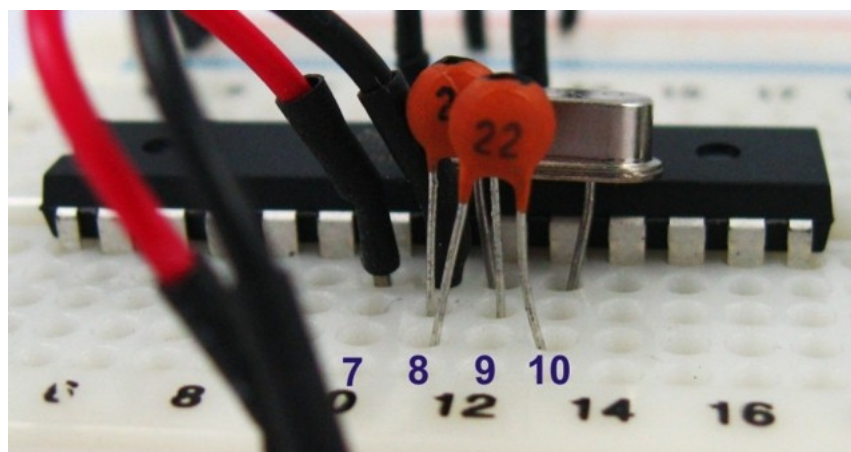
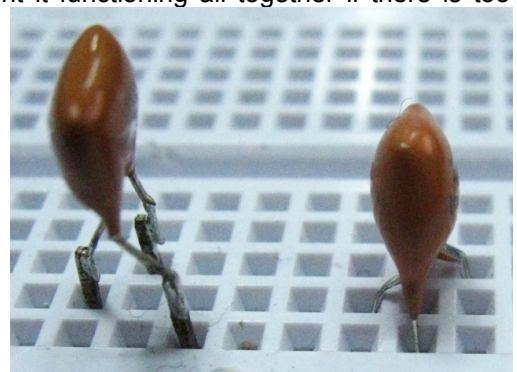
Picaxe offer ready made programming cables that terminate in a 3.5mm stereo plug. A 3.5mm stereo socket can easily be adapted to interface with a breadboard.

Resonators and Crystals:

Originally Mr. General (both Picaxe and Arduino) used a 3 pin, 16MHz ceramic resonator. This is the cheapest and easiest to use. There are 2 ways this can be installed on the breadboard. The first way is to plug it in near the chip and use 3 jumper wires to connect it to the processor. This usually works ok as long as the jumper wires are not too long. The tracks of the breadboard and the wires act like small capacitors and inductors. These can affect the frequency of the clock or prevent it functioning all together if there is too much.

I prefer to bend the legs of the resonator so that it connects directly to the pins of the processor. This is more reliable. As the resonator pins are a bit short it is even better if you solder some extensions onto the pins. Picaxe 28X1, ATmega8, ATmega168 and ATmega328 all have the GND pin (8) and XTAL pins (9, 10) in the same configuration making this easy.

If you are using Arduino then a 16MHz crystal and two 18pF or 22pF capacitors can be used for better timing accuracy. This might work with picaxe although I have never tried it.



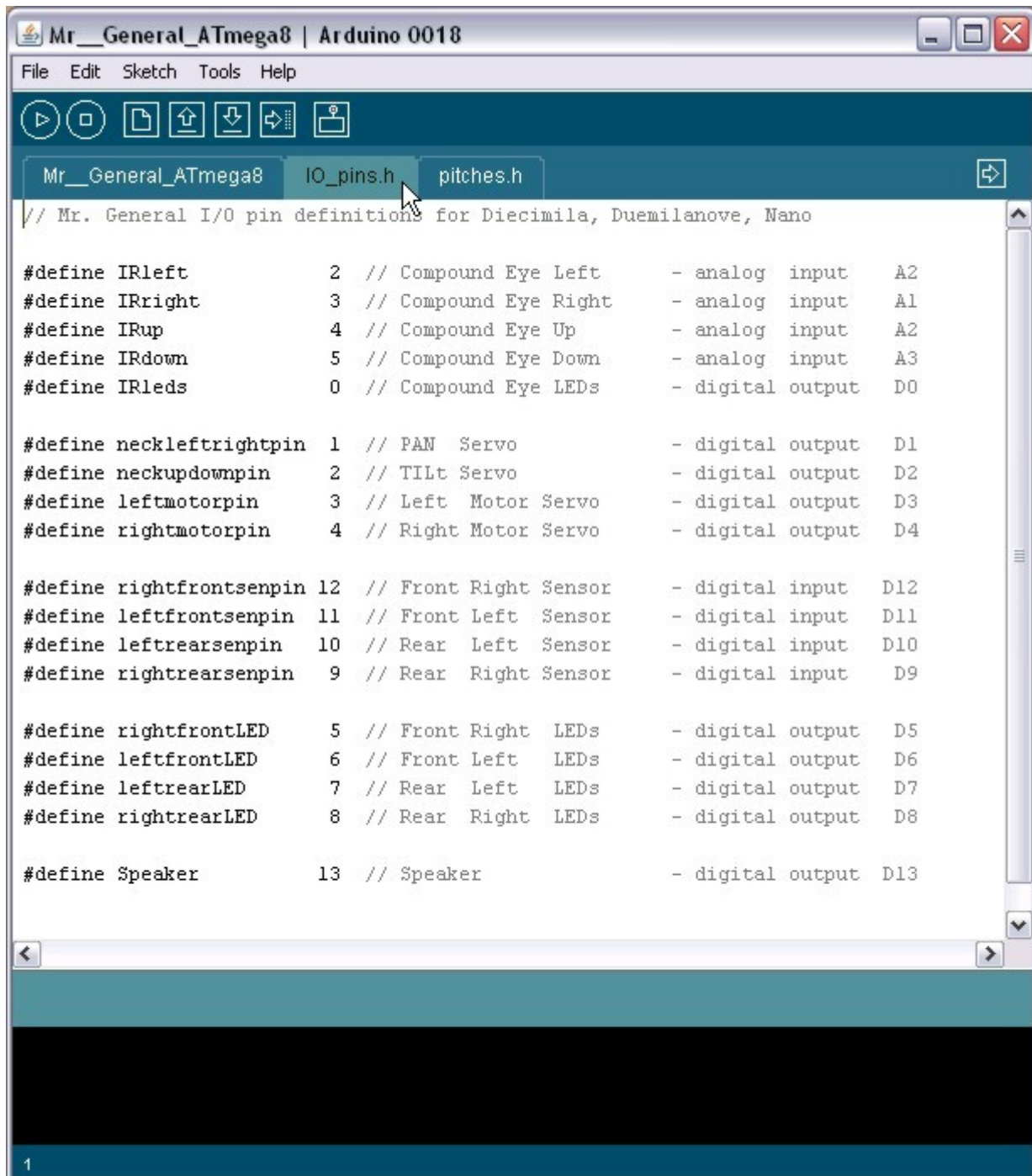
The Code:

The original "Mr. General" kit was designed around the Picaxe 28X1 although many different processors can be used. The program has since been re-written for Arduino and the kit is now normally supplied with the ATmega8A which is equivalent to the old Arduino NG with ATmega8.

If you have an Arduino Nano then this will also plug directly into the breadboard. The Arduino Nano has two extra analog input. This allows the 4 corner sensors to be used as analog sensors.

I have included sample code for ATmega8A, Arduino Nano and Picaxe28X1. If you are using a Picaxe 28X2 or 40X2 then you might be able to use the X2 conversion wizard to convert the program.

Arduino users should note that there is a separate tab called "IO_pins.h" where all the IO pins are defined. This tab is effectively your wiring diagram as it tells you which pin of the processor connects to what piece of hardware on the PCB.



```
// Mr. General I/O pin definitions for Diecimila, Duemilanove, Nano

#define IRleft      2 // Compound Eye Left      - analog input  A2
#define IRright     3 // Compound Eye Right     - analog input  A1
#define IRup        4 // Compound Eye Up        - analog input  A2
#define IRdown      5 // Compound Eye Down      - analog input  A3
#define IRLeds      0 // Compound Eye LEDs      - digital output D0

#define neckleftrightpin 1 // PAN Servo              - digital output D1
#define neckupdownpin   2 // TILT Servo              - digital output D2
#define leftmotorpin    3 // Left Motor Servo        - digital output D3
#define rightmotorpin   4 // Right Motor Servo       - digital output D4

#define rightfrontsenpin 12 // Front Right Sensor      - digital input  D12
#define leftfrontsenpin  11 // Front Left Sensor       - digital input  D11
#define leftrearsenpin   10 // Rear Left Sensor        - digital input  D10
#define rightrearsenpin  9  // Rear Right Sensor       - digital input  D9

#define rightfrontLED    5 // Front Right LEDs        - digital output D5
#define leftfrontLED     6 // Front Left LEDs         - digital output D6
#define leftrearLED      7 // Rear Left LEDs          - digital output D7
#define rightrearLED     8 // Rear Right LEDs         - digital output D8

#define Speaker         13 // Speaker                 - digital output D13
```

Regardless of which processor you use the structure of the sample code is very simple.

The Main loop does these things:

1. Updates the LED pattern approximately 5 times a second. This causes the corner LEDs to chase around the PCB.
2. Updates the position of the neck servos and the speed / direction of the continuous rotation servos.
3. Reads the compound eye.
4. Calculate new pan / tilt servo positions and continuous rotation servo speeds to maintain object tracking.
5. Read the corner sensors and adjust continuous rotation servo speeds to prevent a collision.
6. Optional behaviour routines such as boredom counters can be added here.

Reading the IR sensors:

Simple IR sensors such as those used on Mr. General cannot measure distance accurately and can be tricked by objects of different colours. For example a small bright white object in the distance and a large dark object nearby can give the same readings depending on how well they reflect IR. Sunlight has a lot of infrared light and can easily trick or blind these sensors. Background light such as this is called ambient light.

To help overcome some of these limitations we use the method proposed by Frits of reading the sensors twice. The first time we read the sensor with the IR LEDs on. This gives us a reading that is equal to the ambient light plus IR LED light that has reflected from a nearby object. The second reading is with the LEDs off so that we read only the ambient light. When we subtract the second reading (ambient IR) from the first (ambient IR + reflected IR) we are left with a value that represents only the IR light that was reflected from a nearby object. This technique allows the robot to track your hand movements while ignoring the IR light coming in from a nearby window.

Tracking Motion:

Using the pan servo as an example the robot must first read the left and right compound eye sensors and subtract the ambient light as previously mentioned.

The distance/reflectivity of the object must be taken into account to prevent servo overcorrection. This is done by using a variable called panscale which is the average of the left and right readings divided by LRscalefactor which is adjusted to allow for servo speed and tracking sensitivity. Too sensitive and the servos will overshoot causing them to jitter.

A value called "leftright" is the absolute difference between the left sensor and the right sensor divided by the panscale to allow for distance and servo speed. This value is then added or subtracted from the old servo position to provide a new servo position.

Tilting of the head works the same way as the panning of the neck. Arduino users must forgive the clumsy code as it has been translated from Picaxe basic which has limited math functions and no negative numbers.

Following an object:

In order to track an object properly the robot needs to do more than turn it's head. It must follow with the body. If the neck pans too far left or right then the wheels are driven causing the robot to turn towards the object.

Distance is calculated by averaging the readings of all 4 sensors (up, down, left and right). This is not an accurate measurement of distance but is good enough for the robot to judge things such as which direction is blocked and which direction has room to move. The robot tries to maintain a set distance from the object it's tracking so as not to lose it's lock on the object while avoiding a collision. The wheels are driven forward or backward as required.

Collision avoidance:

Finally the PCB corner sensors are read in the same fashion as the eye sensors. One reading with the LEDs on and one with the LEDs off. This causes all the corner LEDs to pulse briefly. Because of the speed this is happening at the green corner LEDs appear to be dimly lit.

After the readings have been taken any objects detected are compared with the direction of the motors. If necessary a motor will be stopped to avoid a collision.

Light pattern and sensor indication:

If a corner sensor detects an object or the light pattern determines that that corner LED should be on then that LED will be re-lit causing that corner LED to appear bright rather than dim.

Additional behaviour:

In the original picaxe code I also experimented with a boredom counter which monitored how much time had passed without tracking a moving object. If too much time passed then the robot would wander off in a random direction until it found a new object. If that object failed to move in a given time the robot would wander off again.

Add other counters to monitor things such as frustration if the robot gets stuck in a corner. Excitement if the the robot finds a fast moving object or multiple moving objects. By adding several counters and behaviour routines your robot can appear quite life like.

IR Communications:

If a suitable communications protocol was written then it should be possible for more than one robot to communicate with each other using the IR sensors as IR transcievers.

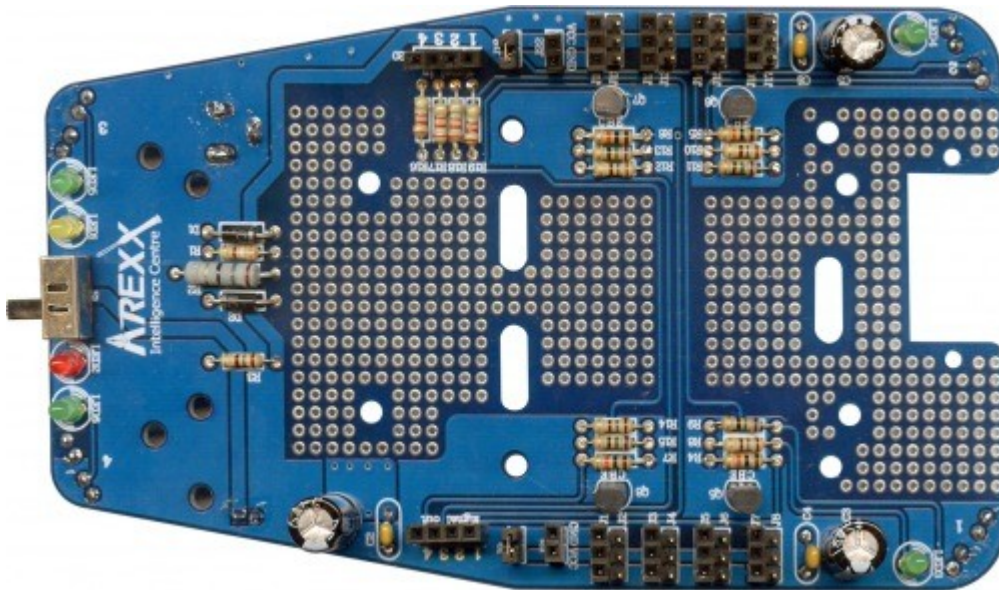
Additional hardware:

The main PCB of Mr. General has all the spare space converted to prototype board. This area is ideal for adding additional circuitry such as line following sensors, [speakjet chipset](http://letsmakerobots.com/node/4305), amplifiers etc. There are 2 great "How to's" for the speakjet here:

<http://letsmakerobots.com/node/4305> this is Ant's post using the picaxe.

<http://letsmakerobots.com/node/13210> this is Droidbuilder post using the Arduino and TTS256 companion processor.

An ideal amplifier is the TDA2822M. This is a small 8 pin stereo power amplifier that will work with supply voltages from 1.8V to 15V. The two amplifiers in the chip can be connected in "bridge mode" to make a single amplifier with a greater power output.



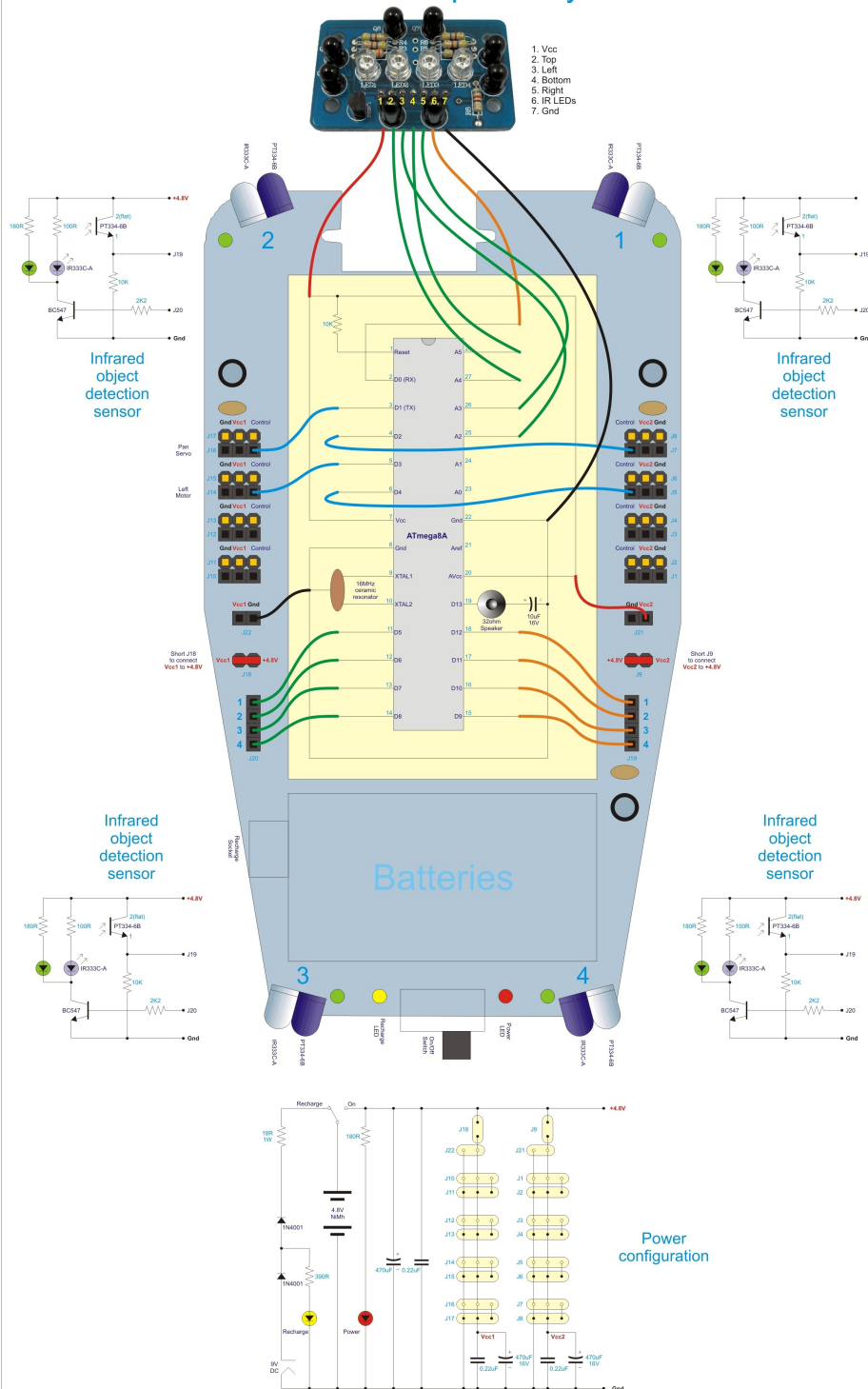
As the continuous rotation servos stop position is sensitive to temperature, a temperature sensor such as an LM35 could be added between the servos. Your processor could use this to adjust the stop position of the servos.

Additional voltages:

You may choose to use 3.3V devices. The area under the breadboard is an ideal location for a 3.3V regulator. As the power rail on each side of the robot can be isolated and changed you could connect the rail on one side of the robot to 3.3V and switch all the servos to the other rail which might remain connected to battery +V.

I hope this guide will help owners of the "Mr. General" robot kit to get more enjoyment from it through a better understanding of it's functions.

1. Vcc
2. Top
3. Left
4. Bottom
5. Right
6. IR LEDs
7. Gnd



1. Vcc
2. Top
3. Left
4. Bottom
5. Right
6. IR LEDs
7. Gnd

