This project is concerned with further developing and completing the ‘Dancing E-Bugs’ open day concept started by David McKechnie in 2008 (Video Based Multi-Vehicle Systems Control over Wireless Networks by David McKechnie [1]) in which a number of wheeled robots are controlled in an interactive manner. This required both software and hardware additions to the existing framework of the system. The robots consist of two CD sized PCB’s and a CD sized ‘bug pattern’ mounted in layers above an aluminum chassis containing two stepper motors. They are battery operated and utilize an Atmel ATMega2561 microprocessor. Control and User Interface are via a desktop or laptop computer running software authored in Visual C# 2008. Images are captured from a webcam mounted on an elevated frame with the MATLAB Image Acquisition toolbox, this allows the software to control the robots in real time, and image processing is also carried out in the MATLAB environment. Wireless communication utilizes XBee OEM RF modules from Digi which are based upon the ZigBee short range wireless communication standard. Communication occurs at 9600 baud.

The second part of the project aimed to use the theory from [3] and the hardware and software from ‘Dancing E-Bugs’ to create a multi-vehicle cooperative control system. The robots, once given a rendezvous point by the PC, communicate with each other and using a consensus algorithm, would work out a common time when they could all meet at the rendezvous point simultaneously. Only preliminary investigation of this topic was achieved due to project timing constraints.
1 Acknowledgements
Primarily I would like to thank my supervisor Dr Ahmet Sekercioglu for his help, advice and understanding throughout this project.
Secondly I would like to thank Mr Ian Reynolds for his help with the E-Bug PCB’s, XBee modules, camera frame and general advice.
Thirdly, my thanks go to Mr Maurice Gay in the Mechanical Workshop for his outstanding help with the mechanical side of the project and the E-Bugs in particular.
Next I would like to thank Mr Ray Cooper for acquiring the materials and parts for this project and being a never ending source of capacitors, resistors, diodes, etc. for it.
Lastly I would like to thank Mr David McKechnie for his general advice and help with C#.
# Table of Contents

1 Acknowledgements .................................................................................. 2
2 Table of Contents .................................................................................. 3
3 List of Figures ....................................................................................... 4
4 Introduction ............................................................................................ 5
  4.1 Aim ................................................................................................... 5
  4.2 Basis for Investigation ................................................................. 5
5 System Summary ................................................................................... 6
  5.1 Overview ....................................................................................... 6
  5.2 Design Assumptions ........................................................................ 6
6 The E-Bug ............................................................................................... 8
  6.1 Hardware ....................................................................................... 8
    6.1.1 Bug Pattern Development ................................................... 9
  6.2 Software ......................................................................................... 10
    6.2.1 Communication Handling ................................................. 12
    6.2.2 Movement ............................................................................ 12
  6.3 Sens-R Robot Faults ......................................................................... 12
    6.3.1 ATmega2561 Fuse Settings .............................................. 12
    6.3.2 Circuit Schematic and PCB design ................................. 13
    6.3.3 5V to 3.3V level conversion ............................................ 14
    6.3.4 Battery Voltage ............................................................... 15
    6.3.5 Stepper Motor Control Software .................................... 16
7 Consensus Algorithm ............................................................................. 17
  7.1 Implementation .............................................................................. 17
8 Wireless Communication ......................................................................... 18
  8.1 Operating Mode ............................................................................ 18
  8.2 Packet Structure ........................................................................... 19
  8.3 Implementation Difficulties ......................................................... 19
9 Colour Detection Software .................................................................... 21
  9.1 findBugColour ............................................................................. 21
10 Control Software ................................................................................. 22
11 Conclusion .......................................................................................... 24
  11.1 Recommendations for further work ..................................... 24
12 References ........................................................................................... 25
13 Appendices ......................................................................................... 26
  13.1 Parts List ..................................................................................... 26
  13.2 E-Bug Source Code (C) ............................................................. 27
  13.3 Control Software Source Code (Visual C#) ......................... 33
  13.4 MATLAB Source Code ............................................................ 51
  13.5 E-Bug Logic Circuit Schematic ............................................. 56
  13.6 E-Bug Logic Circuit PCB ......................................................... 57
  13.7 E-Bug Power Circuit Schematic ............................................. 58
  13.8 E-Bug Power PCB ................................................................. 59
3 List of Figures

Figure 1: 'Dancing E-Bug' system overview [1] ................................................................. 7
Figure 2: The E-Bug ........................................................................................................... 8
Figure 3: Old E-Bug "Bug Pattern" .................................................................................. 9
Figure 4: Bug Pattern radial stripe at 40%, 50%, 60%, 75% and 90% opaqueness .......... 9
Figure 5: New E-Bug “Bug Pattern” ............................................................................... 10
Figure 6: AVRISP mkII with 10 pin header added............................................................. 10
Figure 7: E-Bug Software Overview ............................................................................... 11
Figure 8: USART data signal with incorrect fuse settings (Yellow) and correct settings (Blue). 13
Figure 9: Old and new PCB designs .............................................................................. 14
Figure 10: Signal on XBee Data In pin using signal diode .............................................. 15
Figure 11: Signal on XBee Data In pin using Schottky diode ......................................... 15
Figure 12: XBee Series 2 RF Transceiver on USB serial explorer board ....................... 18
Figure 13: New and Old Packet Structures .................................................................... 19
Figure 14: Control Software and User Interface Overview [1] ........................................ 22
Figure 15: E-Bug Control GUI Run! Window ................................................................. 23
4 Introduction

4.1 Aim

There were two aims for this project, the primary aim was to complete the ‘Dancing E-Bugs’ open day concept and the other was to implement multi-vehicle cooperative control using the Dancing E-Bug system and an information consensus algorithm.

The primary aim consisted of further developing and completing the work started by David McKechnie in his undergraduate project, Video Based Multi-Vehicle Systems Control over Wireless Networks[1]. Specifically this involved

- Developing a colour identification algorithm in MATLAB to allow the software to distinguish between different E-Bugs using colour
- Developing a new E-Bug robot and three copies
- Writing software to operate the new E-bugs
- Completing the wireless communication link between the base station and E-Bugs using XBee wireless units
- Modifying the existing control software to be capable of operating with multiple E-Bugs
- Completing the command auto-queuing capability of the control software
- Integration of the existing framework with the new camera setup

The secondary aim of the project is to utilise the theory from [3] and the hardware and software from ‘Dancing E-Bugs’ to build the basic framework for a multi-vehicle cooperative control system. The E-Bugs, once given a rendezvous point by the PC, will communicate with each other over the wireless network, and using a consensus algorithm will try to work out a common time when they can all meet at the rendezvous point simultaneously.

4.2 Basis for Investigation

The system is designed to showcase many of the areas of study found in an undergraduate ECSE course at Monash University. Such as:

- Engineering Software Design (MATLAB – Image acquisition and processing)
- Wireless Communications (XBee RF modules)
- Embedded software design (C)
- Desktop software design (Visual C#)
- Switching electronics
- Control algorithms and theory

As it demonstrates such a wide range of topics and shows many of the skills learnt in the ECSE undergraduate course it is an excellent project for an Open Day presentation. It is designed to present the concepts in an interesting and interactive manner allowing easy interaction with Open Day attendees.
5 System Summary

5.1 Overview
The system is composed of 3 main sub-systems which are most easily separated by programming language. These systems are:

1. The E-Bugs: This includes software written in C for the ATmega2561 to handle communication and control the stepper motors, as well as the hardware design. The E-Bugs are based on a robot developed in [2] with large modifications made by the author. They are created from a stack of two CD sized PCB’s and a ‘bug pattern’ top layer all on top of an aluminum chassis housing two 10V bi-polar stepper motors. They are powered by 9 rechargeable 1.2V ‘AAA’ NiMH batteries providing 10.8 volts and also have an inbuilt charger. Wireless communication is achieved with XBee RF Transceiver units.

2. The image processing system: Written in MATLAB by David McKechnie[1] and modified by the author, this part of the system takes data from a USB webcam mounted on an elevated frame, processes it and provides an output of the co-ordinates of any bug like shape detected, the colour, the radius of each shape and the degree of rotation.

3. PC communications, control and user interface software: Written in Visual C# 2008 by David McKechnie[1] and modified by the author. Communication with MATLAB is via the Component Object Model interface available for MATLAB. Both software items are run concurrently, with the MATLAB server slaved to the C# software, which executes commands remotely. The software makes extensive use of Windows .Net runtime libraries. It handles communication between the user and the E-Bugs and creates a graphical user interface (GUI) to make user interaction with the system easier.

An overview of the system is presented in Figure 1.

5.2 Design Assumptions
As the project utilises and continues the work of others and because of the limited time frame of the project, several design assumptions were made. They are as follows:

- The software specified in the project Dancing E-Bugs: Video Based Multi-Vehicle Systems Control over Wireless Networks[1] by David McKechnie works as stated in the final report.
- The robot designed and built by Aidan Galt for the project Design of an Improved Sens-R CPU board[2] works as stated in the final report and is fit for the purpose.
Figure 1: 'Dancing E-Bug' system overview [1]
6 The E-Bug

The final E-Bug consists of 2 10V bi-polar stepper motors with lock nutted wheels attached to a 2mm sheet aluminium frame pressed into shape with plastic ball castors on columns at the front and back providing support. The frame supports the power supply which consists of 9 1.2V batteries connected in series to produce 10.8V above which in layers, from the bottom up, are the power PCB, logic PCB and bug pattern. Wireless communication is handled by a XBee RF transceiver module operating on the 2.4GHz spectrum. The robot is controlled by a program written in C running on an Atmel ATmega2561 microprocessor.

6.1 Hardware

The main E-Bug hardware design is taken from [2], with heavy modification. The optical mouse situated at the bottom of the robot has been removed along with the associated wiring and supporting aluminium frame. Additionally the recharging docking station has been removed and the wheels have been shifted as close to the chassis as possible. This was done to remove redundant components, reduce the considerable weight of the E-Bug and to streamline the look and shape.

Two universal direction plastic castor wheels on plastic columns have been added to the E-Bug, one at the front and one at the back. With the removal of the optical mouse the E-Bug no longer balanced, so the castors were added to keep the robot stable. Castors made out of plastic were chosen as they are cheap, very light and produce very little friction and could therefore be added to the E-Bug without any significant increase in cost, weight or reduction in battery life.

A “bug pattern” top layer has been added to the E-Bug so that it can interface with the bug finding algorithm developed by in [1]. This is simply a 12cm diameter, CD sized, piece of paper with a red circle and a 3cm wide, 85% opacity radial stripe printed on it, glued to a plastic disk and attached above the top (logic) PCB of the E-Bug.

A spike and lock nut mechanism has been added to the axel of each stepper motor to attach the wheels to the axel. This was done to ensure that the wheels were robustly attached and to maintain accuracy of steps by eliminating slip between each stepper motor and its wheel.
6.1.1 Bug Pattern Development

The original bug pattern developed in [1], a circle with a 1.5cm wide, 40% opacity radial stripe, proved to be unsuitable for the new camera setup. The radial stripe was too opaque and too thin to be easily distinguished, especially in varying light conditions, from the background of the bug pattern by the image processing software in the down sampled web camera image. It was consistently unobservable and tended to fade in and out of the white background of the bug pattern.

The first attempt to solve this issue was to create a series of new bug patterns with increasing radial stripe opacity. Starting at 50%, the opacity was increased in 10% increments to 90% with each level of opacity being tested under the webcam. However none of these opacity levels were consistently observable. No matter what level of opacity the stripe was it still became indistinguishable from the white background too often to be reliable.

The second attempt to solve this issue was to use the same procedure as attempt 1 but to use a stripe of double the thickness (3cm). Starting at 50% opacity and increasing to 90% in 10% increments again, it was found that 85% opacity with the 3cm stripe provided the most reliable detection while still being opaque enough to not register during the white threshold algorithm. Detection is still not 100% reliable when lighting conditions change suddenly but it is believed that this is because the autofocus feature of the webcam is turned off, as if it is refocused after a change in lighting, the stripe is picked up again.
6.2 Software
The software for the E-Bug has been written using the AVR Studio 4 development environment with the WinAVR GNU GCC implementation for compilation. The board is programmed using an Atmel AVRISP mkII, using a 10 wire In System Programmer (ISP) [6]. Modifications were required to the new AVRISP mkII purchased for this project as it possessed only a 6 wire header. These modifications are documented in [7]. It is required to use the AVRISP mkII as this is what the E-Bugs are designed to be programmed with and it provides a cheap, reliable and easy to use connection to the ATmega2561.

As the E-Bug requires the ability to handle wireless communication while simultaneously executing movement commands two options for the software were identified. The first was to utilise the multi thread capabilities of a Real Time Operating System (RTOS) and the second option was to use a normal single threaded program with communication handled by interrupt service routines.

As the author has prior experience with µC/OS II and it was utilised successfully in [1] it was originally planned to utilise the RTOS option. However due to difficulties during development due to the, lack of, port files required to run uC/OS II on the ATmega2561 and the time constraints of the project this option was eventually abandoned.

Therefore the alternative of a sequential C program with communication handled by interrupt service routines was developed.
Figure 7 shows the basic layout of the E-Bug software. The program initially calls a function which initialises the microprocessors USART for serial data transmission and reception. The program then waits on a 5 second delay to ensure that the XBee module has connected to the network before entering an infinite loop that handles movement of the E-Bug. Data reception is handled by an interrupt service routine.
6.2.1 Communication Handling
Reception is handled by an interrupt service routine that is triggered by \texttt{USART1_RX_vect}. Each incoming data byte is checked to see if it is the start byte, preamble, for a new packet before it is stored in the rxBuf array. If the start of a new packet is detected the byte is shifted to the front of the array and rxBufPtr is set to zero. Each time a data byte is detected rxBufPtr is incremented. Once more than 5 bytes have been detected after the preamble it is assumed a whole packet has been received and the destination and source addresses are checked. If the packet is for the E-Bug a flag is set indicating that a valid packet has been received.
The received packets sequence number is then checked inside the main program and if it is correct the contents of the packet is removed and the appropriate movement/rotation flags are set.

Transmission, only required to indicate completion of a command, is executed by the send_pkt function which prepares the packet and then activates the transmission interrupt service routine which sends the packets to the USART.

6.2.2 Movement
Movement is handled by the polling loop inside main. Each iteration checks the status of the two movement flags rotateFlag and fwdFlag. If a rotate flag is detected the appropriate excitation sequence is selected for the desired direction of rotation. Then a for loop is entered that iterates the number of steps required to rotate the specified angle. The number of iterations is determined by the \texttt{angle} lookup table.
Forward movement is achieved in a similar manner to rotation in that the required excitation sequence for the motors is set and a for loop is entered that iterates an amount determined by the \texttt{forward} lookup table.
Once a command has been successfully completed a \texttt{RDY} (ready) packet is transmitted back to the base station to indicate successful completion of the command and the robots availability for a new command.
At the end of each iteration of the for loop a check on the flag status is performed to see if a new packet has arrived that invalidates the current command. If it has, movement is ceased until the new packet has been decoded. In this way new commands can be issued to an E-Bug to overwrite currently executing commands.

6.3 Sens-R Robot Faults
There were a number of shortcomings present in the robot designed in [2] causing it to fail its design assumption listed in section 5.2 Design Assumptions. These were all due to vastly inadequate testing, poor record keeping and failure to update documentation during the course of or at the end of the project. This had a large knock on effect on this project and consequently major delays were experienced. The following is a list of faults, in the order encountered, and the steps taken to correct them.

6.3.1 ATmega2561 Fuse Settings
During early testing of the robot developed in [2] a simple program was written to test the wireless interface. It simply transmitted a byte from the microprocessor to the XBee module four times a second. During testing it was noticed that the frequency of transmission and period of bytes being sent from the microprocessor did not match the set baud rate of 9600 baud. Upon investigation and after eliminating other possible sources of failure it was discovered that the fuse settings for the ATmega2561 given in [2] were incorrect. The settings given did not set the processors clock to the external crystal but instead had it set to use the processors inaccurate internal clock. This also meant that the clock frequency specified in the software did not match the actual clock frequency being used by the device.
Investigation of the datasheet revealed the correct fuse settings to enable use of the external crystal oscillator and once these were applied the expected output was provided by the microprocessor.

### 6.3.2 Circuit Schematic and PCB design

Early on it was discovered that the circuit schematic for both the logic and power circuits did not accurately represent the actual circuit design of the robot. While the wiring and active components were correct the values given for the passive components were not. It is not known whether these were simply never changed from the default values that appeared when the diagram was created or if these were values trialled during the development of the robot and not subsequently updated at the end of the project. In any case this required a painstaking process of identifying capacitor and resistor values by hand from their codes and colours and updating the schematic with the correct values.

As the project required 3 robots and only a single prototype existed at the start of the project several copies of the logic and power PCB’s needed to be ordered. However before the PCB’s could be ordered changes, due to the removal of the optical mouse and its associated wiring from the robot, needed to be made to the existing design. Additionally the inaccuracy of the circuit schematic prompted a closer study of the PCB design from [2].

Upon investigation of the PCB design it was discovered that a net list had not been used to assist track layout and display connections between circuit elements. Instead the tracks had been laid manually and this had resulted in a large number of poor connections between tracks and components. Additionally it was found that many components had incorrect hole sizes and several had been laid out backwards. Like the circuit schematic, the PCB design had not been updated when these mistakes were discovered during the development of [2].

Correcting this required updating the PCB design with the correct dimensions and orientation of each component sourced from its mechanical drawing in its datasheet. Design rule checks were run in Altium, using the limitations provided by Futurlec the manufacturer of the PCB’s, until the design passed without errors or warnings.
6.3.3 5V to 3.3V level conversion

The XBee RF transceiver modules used for wireless communication between the E-Bug and the base station are not 5V tolerant and therefore require all input signals from the ATMega2561, running at 5V, to be level shifted to 3.3V. As there is only a single input, Data In, to the XBee module from the microprocessor this was achieved in [2] through the use of an in series signal diode and pull up resistor. This arrangement appeared functional during initial testing as wireless communication was achieved. The XBee module received a modulated signal that varied between 0.65V for a low and 3.3V for a high.

However a short time after wireless communication was achieved it failed. Even after replacing the signal diode, pull up resistor and XBee module wireless communication could not be restored. It was then decided that given the troublesome nature of the etched PCB used in the prototype, work on the problem would be discontinued until the new factory made PCB’s arrived. It was thought that with the better production values of these PCB’s the problem would be resolved however this was not the case.

Upon further investigation it was realised that the voltage being received by the XBee module was not low enough to register a logic zero. According to the XBee datasheet a logic low must be less than 0.2 * Vcc, which in this case was

\[ Logic\ low\ =\ 0.2 \times 3.3\ V = 0.66\ V \]

However the voltage drop across a standard signal diode is 0.7V – 0.8V so the XBee was receiving a voltage of approximately 0.7V for a logic low, too high to register as a logic low. Wireless
communication had only worked in the first instance because the voltage drop across the diode was just
low enough and/or the XBee was sensitive enough to register the voltage as a logic low.

![Figure 10: Signal on XBee Data In pin using signal diode](image1)

To correct this fault, the signal diode was replaced with a schottky diode which has a lower voltage drop,
in the order of 0.3 – 0.4V, making a logic low now be at approximately 0.3V, well below the 0.66V
threshold. While correcting this problem did not fix wireless communication it was one of several
problems contributing to the failure of that mechanism.

![Figure 11: Signal on XBee Data In pin using Schottky diode](image2)

### 6.3.4 Battery Voltage

Once wireless communication was achieved and full scale testing of the system commenced it was
discovered that the stepper motors would only run for very short periods of time after which they would
simply produce noise when power was applied to them. After consultation of the motor datasheet it was
realised that the battery voltage of the robot was insufficient to drive the stepper motors. When fully
charged the batteries would reach a peak of approximately 10.5V, sufficient to drive the motors, before
quickly falling to their insufficient nominal value of 9.6V after a short period of use. This problem was corrected by adding a ninth battery to the robot to boost the battery voltage to a nominal 10.8V. The removal of half the batteries and addition of a boost converter was in development but unable to be completed by the time of writing.

6.3.5 Stepper Motor Control Software
The attachment of a wheel to the axel of each stepper motor was achieved in [2] through the use of a drill and glad wrap. This did not produce a strong attachment and a large amount of slippage occurred during movement. To correct this problem a spike was put through the axel which slotted into a groove cut into the inside of each wheel and a locknut was attached to the outside to keep the assembly in place. This prevented any slippage occurring between the axel and wheel.
However this also revealed that the control software developed in [2] for the stepper motors was not correct. While it managed to drive the motors when there was large amounts of slip once this had been removed the drive sequence being sent by the software was no longer able to drive the motors. Investigation revealed that the sequence developed in [2] was not energising the coils inside each stepper motor in the correct sequence so as to produce “full wave” drive. This was corrected by changing the sequence used to energise the coils of each motor.
7 Consensus Algorithm

The basic idea of a consensus algorithm is for each vehicle in the system to communicate its information state with the other vehicles in the system in order for all the states of the team to come into agreement [4]. The actual mathematical algorithm specifies the dynamics used to determine that agreement. One of the most common consensus algorithms is given by:

\[ \dot{x}_i(t) = - \sum_{j=1}^{n} a_{ij}(t) (x_i(t) - x_j(t)), \quad i = 1, \ldots, n, \]  

(1)

However as communication between vehicles will occur at discrete instants of time I will be using a difference equation, the most common of which is given by:

\[ x_i[k+1] = \sum_{j=1}^{n} a_{ij} x_j[k], \quad i = 1, \ldots, n, \]  

(S3)

However a critical question with these algorithms is when do all of the vehicles converge to a common value? The answer is that while they ensure that the information states of the team come into agreement, they do not give a specific common value [4]. What this means is that while the algorithms guarantee that every member of the team reaches a common negotiated value, they do not specify what that value is. Consensus is reached is when:

\[ \dot{i}, \ j = 1, \ldots, n; \ |x_i[k] - x_j[k]| \to 0 \text{ as } k \to \infty \]

See [4] for more information.

7.1 Implementation

The PC will give the E-Bugs their rendezvous point and emulate the localisation module in each E-Bug by giving them their position coordinates, distorted by random errors. The individual E-Bugs are then responsible for communicating with each other and implementing the consensus algorithm to negotiate a simultaneous rendezvous time.

Unfortunately delays in the project due to development of the E-Bug and implementation of wireless communication meant that only preliminary investigation of this topic was achieved.
8 Wireless Communication

Wireless communication in the system is handled by Digi XBee OEM Series 2 RF Transceiver modules, pictured in figure 12. On the computer side a XBee module is mounted on a XBee Explorer Board with a USB connection to the computer. This allows direct access to the serial and programming pins on the XBee module. This board is also used to configure the parameters for the E-Bug XBee modules. On the E-Bug side, the DIN and DOUT pins of the XBee module are connected directly to USART1 on the ATmega2561. The XBee modules handle packetisation of data, retries and acknowledgements internally removing the need for this to be done in software. Transmission is much more reliable than that of the packet radio units used in [1] which allowed for simplification of the software that handles communication on both the computer and E-Bug sides.

8.1 Operating Mode

The XBee modules operate in Transparent Mode which causes the modules to act as serial line replacements. All data sent to a module via the DIN pin is queued up for transmission. When data is received, the data is sent out through the DOUT pin. The module configuration parameters are specified using the AT command mode interface.

Data to be transmitted is held in the serial receive buffer until one of the following causes the data to be packetized and transmitted:

1. No serial characters are received for the amount of time determined by the RO (Packetization Timeout) parameter.
2. Maximum number of characters that will fit in an RF packet is received (72 bytes).
3. The Command Mode Sequence (GT + CC + GT) is received. Any character buffered in the serial receive buffer before the sequence is transmitted.

As the amount of data being sent is small data transmission is almost always triggered by case 1 when there is a lull between data being loaded into the XBee’s serial buffer. RO is set to the default value of 3ms.

Transparent mode was chosen at it is the simplest and easiest way of interfacing the modules with their respective hosts. The alternative mode, API Operation, provides much greater functionality but is also much more complicated. As the advanced features of API Operation are not required for this project it was decided to implement the simpler solution of Transparent Mode.

The XBee modules utilise the ZigBee protocol which is based upon the 802.15.4 protocol. For more information see [10].
8.2 Packet Structure

The packet structure is deliberately similar to that used in [1]. As this project built upon the work of [1] it made sense to utilise the same packet structure rather than re-inventing it. The only change to the packet structure in [1] is the removal of the CRC8 checksum as error correction is handled by the XBee modules and therefore does not need to be implemented in software.

In addition to removing the checksum from the packet structure the ACK function has also been removed as it is now a redundant feature. As the XBee’s handle ACK’s internally between themselves when a packet is sent it can be assumed that the transmitted packet reaches its destination. There are only three cases where data transmission between XBee units will fail:

1. The destination XBee unit is receiving so much data that its serial receive buffer reaches capacity and blocks incoming data
2. The channel is so noisy or so full that data transmission is entirely blocked
3. The destination XBee module is out of range of the transmitting module

Option 1 is very unlikely to occur as packets contain little data, frequency of transmission is low and only 4 XBee units exist in the network therefore bandwidth utilisation is quite small even at peak times and is much less than the maximum capability of the XBee units.

Option 2 is unlikely as the XBee’s have a wide range of channels to select from and select the least noisy channel they can find upon network startup. Additionally as only 4 XBee units exist in the network transmission being blocked due to maximum capacity being reached is improbable.

Lastly, as communication only occurs between the coordinator and end nodes, if the destination module is out of range then the sending unit itself must be out of range and will therefore no longer be in the network.

8.3 Implementation Difficulties

During development of the wireless network to be used for communication between computer and E-Bugs several problems were encountered. Most of these were hardware related and are listed under The E-Bugs->Hardware->Sens-R Robot faults however a particular problem related to the XBee units themselves. Once the correct signal was being output by the ATmega2561, was stepped down to the correct voltage and the correct settings had been applied to each XBee module bi-directional wireless communication was not functioning. The XBee modules had formed a network, registered the presence of each other and the E-Bug could receive data from the base station coordinator XBee module however serial data sent to the XBee on the E-Bug for transmission to the base station failed to transmit. Bi-directional communication could be achieved using a chatroom like protocol when two XBee units were
placed in explorer boards attached to computers but the data being sent from the simple transmit infinite program running on the E-Bug’s microprocessor failed to transmit.

It was a most frustrating problem as the correct signal was being sent to the XBee modules and all literature, including the Digi tech support wizards, said that it should be working. Investigation and correction of this problem caused a major delay of approximately 5 weeks to the project schedule. It was eventually realised after strenuous investigation that if data was transmitted to the XBee modules Data In port before the module had associated with the network that it would not register the start bit of a data byte. For unknown reasons the XBee’s serial port could not recover from this error and would not recognise data bytes from that point onwards. This problem was corrected by instituting a 5 second delay at the start of all programs utilising the XBee modules to allow time for network association to occur.
9 Colour Detection Software

This is an addition to the image processing system developed in [1]. While the software in [1] detects E-Bugs with great reliability and returns their x, y coordinates and angle of rotation it has no way of keeping data matched to a specific E-Bug or distinguishing between different E-Bugs. What this means is that if two E-Bugs were to swap positions the software in [1] could not tell the difference and would believe it was the same bug. This becomes a problem when using the automated position controls when more than one E-Bug is present as the software does not know which set of detected coordinates belongs to which bug. To use this feature the software needs to be able to relate the position information received from bugfinder3.m of the detected bugs to specific bugs so it can keep track of each bug as it moves around the image.

9.1 findBugColour

To achieve this, a colour detection algorithm was implemented so that each E-Bug could be distinguished by the colour of its bug pattern, as seen in figure15. This was achieved in the following way:

I. What colours to use? As the camera image uses the RGB format and the system consists of 3 bugs it was a logical choice to make one E-Bug red, one green and one blue thus providing the greatest distance in colour spectrum between the colour of each E-Bug.

II. How to find the colour on each bug pattern? The software that finds the dimensions of each object detected to determine if it is a bug or not does so by finding the first black pixel in the threshold image on the axis of its line of search from the centre of the detected object. Due to the threshold level this first black pixel must be a pixel on the coloured circle around the edge of each bug pattern. The location of each detected black pixel is looked at on the full colour image and the RGB value for this location is stored in the pix_colour array.

III. How to determine the colour? The RGB values for the 3 detected pixels are passed to findBugColour which sums and averages the Red, Green and Blue values. The averages for each colour are the compared and whichever colour has the highest value is the colour of the bug.

IV. How to pass to C#? The 3 colours are assigned a numerical value (Red = 0, Green = 1, Blue = 2) and this is returned to bugfinder3.m. If a colour cannot be determined a -1 is returned indicating identification failure. bugfinder3.m adds the value returned by findBugColour to the array that contains the x, y coordinates and angle values that it passes back to the C# environment.

Providing the correct threshold level is specified by the user this algorithm is quite robust in its detection mainly due to the large difference in RGB values between the bug pattern colours chosen. Even in poor lighting conditions or when a pixel is on the border between two different colours the colour of the bug pattern circle will show through enough to tip the value of its colour higher than the others and ensure detection.

The algorithm only fails when the white threshold has been so incorrectly specified by the user that not a single pixel recorded by the algorithm is the correct colour.
10 Control Software

The base station Control software and user interface was written in C# by David McKechnie for [1] and while it has been modified somewhat during this project it is still largely the same. Therefore it will not be described in detail here but an overview can be seen in figure 14.

![Control Software and User Interface Overview](image.png)

The modifications made are listed below.

1. `serialPort_DataReceived`: Due to the replacement of the Monash packet radios used in [1] with the XBee RF Transceiver modules the serial communication software was slightly altered. As described in 8: Wireless Communication the XBee modules make use of a checksum and ACK’s redundant so use of these has been removed.

2. `bgWrkerGetCoords`: With the colour of each detected bug being given as well as its coordinates and angle of rotation the function was updated to fetch the extra data from MATLAB and then assign the fetched coordinates to the correct E-Bug in the `bugs[]` array as determined by its colour.
   a. If the colour received from MATLAB is -1, indicating a detection failure, the corresponding coordinates and angle are discarded to avoid incorrect data being assigned to any E-Bug. This is done in the hope that the bug will be detected correctly on the next cycle.
3. *buttonSetBugs_Click:* When the user selects the number of bugs to be used each E-Bug is arbitrarily assigned a colour starting with red then green and finally blue. In this way each bug is initiated with a colour so that data can be correctly assigned by *bgWrkerGetCoords*. The limitation to this is that if only one E-Bug is to be used it must be the red E-Bug and if two are to be used it must be the red and green ones and so on.

4. *panelBugSpace_Paint:* To differentiate between multiple bugs in the GUI they must all be drawn different colours. This function was changed so that each E-Bug is drawn in its correct colour in the detected position plot window. This can be seen in figure 15.

5. *Quit_Click:* A quit button was added to the GUI to provide a means of clean exit from the program.

![Figure 15: E-Bug Control GUI Run! Window](image)

6. *bgWrkerAutoControl:* This is the Automatic Control function that allows the user to direct a particular E-Bug to a point on the screen using the green cross hairs but it was not developed to a functional state in [1]. By implementing the command queue using Array Lists this function was developed to a reliable working level.
   a. *buttonAutoGo_Click:* Linked to *bgWrkerAutoControl* this function was changed to use Array Lists so as to be compatible with that function.

7. *buttonSquare_Click:* Like *bgWrkerAutoControl* this function, designed to make an E-Bug move through a 100x100 square infinitely, was not successfully completed in [1]. Array Lists were again used to develop this function to a reliable level.
11 Conclusion

The main aim of this project, to complete the Dancing E-Bugs system, was thoroughly investigated and appropriate hardware and software developed. Areas of investigation included:

- Introduction of a functional multiple access wireless communication system
- Modification and integration of the robot developed in [2]
- A method to differentiate between individual E-Bugs
- Programming a multi threaded windows form application
- Modification of [1] for use with multiple robots
- Development of [1] and [2] into a complete system

In conclusion this project successfully delivered:

- 3 working E-Bugs, including software for communication and movement
- A multiple access wireless communication system
- Software capable of identifying each E-Bug by its colour
- A complete working Dancing E-Bugs system

Due to time constraints and prolonged delays in the development of the Dancing E-Bug system only preliminary investigation of Consensus Algorithms was achieved. These delays were mainly caused by the projects this work utilised not performing as assumed in section 5.2 Design Assumptions. If the previous works had performed as stated the Dancing E-Bug system would have been finished earlier and much more progress on multi-vehicle cooperative control using consensus algorithms would have been made.

11.1 Recommendations for further work

The “Dancing E-Bug” system is currently in a complete form without the scope for much further work. Implementation of a RTOS on the E-Bugs to increase efficiency and reaction time and stream lining of the C# code are possible as minor house keeping improvements. Addition of a boost converter to the E-Bug to replace batteries and incorporate more switching electronics elements is also a possibility as is packaging all the different elements of the base station code into a single executable program that can be run on any laptop or PC without the need of having MATLAB or Visual C#.

As a base for other multi-vehicle based projects there are a myriad opportunities for further study. Areas such as mutli-vehicle cooperative control using consensus algorithms, which was investigated slightly during this project, are one possibility as is wireless mesh or sensor networks.
12 References


13 Appendices

13.1 Parts List

Several major components have been purchased and many more will need to be purchased to complete this project. Small components and miscellaneous supplies such as wires and solder have not been included. The components are listed in the table below:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Description</th>
<th>Part code</th>
<th>Supplier</th>
<th>Per unit cost (retail)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Ball Castor</td>
<td></td>
<td>Pololu</td>
<td>$3.99</td>
</tr>
<tr>
<td>1</td>
<td>AVRISP mkII programmer</td>
<td></td>
<td>Avnet</td>
<td>$54.21</td>
</tr>
<tr>
<td>1</td>
<td>USB Xbee Explorer Board</td>
<td>WIR-045</td>
<td>Ocean Controls</td>
<td>$35.75</td>
</tr>
<tr>
<td>4</td>
<td>Xbee RF module</td>
<td>WIR-041</td>
<td>Ocean Controls</td>
<td>$49.50</td>
</tr>
<tr>
<td>1</td>
<td>USB male A to mini USB cable</td>
<td>USB-003</td>
<td>Ocean Controls</td>
<td>$9.96</td>
</tr>
<tr>
<td>3</td>
<td>Atmega2561 microcontroller</td>
<td></td>
<td>Digikey</td>
<td>$15.91</td>
</tr>
<tr>
<td>3</td>
<td>16x2 character LCD module</td>
<td>Z7013</td>
<td>Altronics</td>
<td>$27.00</td>
</tr>
<tr>
<td>3</td>
<td>7805 - 5V voltage regulator</td>
<td>Z0505</td>
<td>Altronics</td>
<td>$1.20</td>
</tr>
<tr>
<td>3</td>
<td>LM3940 - 3.3V voltage regulator</td>
<td>ZV1565</td>
<td>Jaycar</td>
<td>$2.95</td>
</tr>
<tr>
<td>3</td>
<td>LM317 - variable voltage regulator</td>
<td>Z0545</td>
<td>Altronics</td>
<td>$2.10</td>
</tr>
<tr>
<td>27</td>
<td>2N7000 MOSFET</td>
<td>Z1555</td>
<td>Altronics</td>
<td>$0.47</td>
</tr>
<tr>
<td>16</td>
<td>AAA’ Rechargeable NiMH 1.2V Batteries</td>
<td>SB1723</td>
<td>Jaycar</td>
<td>$3.50</td>
</tr>
<tr>
<td>4</td>
<td>4x’AAA’ battery holders</td>
<td>PH9268</td>
<td>Jaycar</td>
<td>$1.85</td>
</tr>
<tr>
<td>6</td>
<td>MOT-110 Bi-Polar Stepper Motors</td>
<td>MOT-110</td>
<td>Ocean Controls</td>
<td>$22.00</td>
</tr>
<tr>
<td>2</td>
<td>11.059200 MHz crystal</td>
<td>V1265</td>
<td>Altronics</td>
<td>$3.50</td>
</tr>
<tr>
<td>6</td>
<td>10k trimpot</td>
<td>R2361</td>
<td>Altronics</td>
<td>$2.20</td>
</tr>
<tr>
<td>6</td>
<td>L293D Quad Half-H Driver</td>
<td>Z2900</td>
<td>Altronics</td>
<td>$7.95</td>
</tr>
<tr>
<td>3</td>
<td>LM393 - Dual voltage comparator</td>
<td>Z2520</td>
<td>Altronics</td>
<td>$2.80</td>
</tr>
<tr>
<td>3</td>
<td>Pair of wheels</td>
<td>00420</td>
<td>Sparkfun</td>
<td>$6.95</td>
</tr>
</tbody>
</table>
13.2 E-Bug Source Code (C)
/********************
Author: Rory Paltridge
Date: 23/8/2009
Name: EBug_sequential
Version: 3
Status: Working
Description: This program is the overall E-Bug program implemented using a sequential C program and interrupts
Problems:
- Lookup table needs to be tuned
- Need to check destination address of packet in ISR so as not to reset movement flags for commands that are not meant for us.... DONE
Changes:
*********************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#ifndef F_CPU
// define cpu clock speed if not defined
#define F_CPU 11059200 // external crystal
#endif
// set desired baud rate
#define BAUDRATE 9600 // 19200
// calculate UBRR value
#define UBRRVAL ((F_CPU/(BAUDRATE*16UL))-1)
#define ADDRESS 0x00 // Address of node
#define ACKBYTE 0x10
#define RDYBYTE 0x11
#define BUFFSIZE 50
#define PREAMBLE 0xAA // Preamble byte
#define FORWARD 0xA
#define PIVOTRIGHT 0x6
#define PIVOTLEFT 0x9
volatile int rxBuf[BUFFSIZE];
volatile int rxBufPtr = 0;
volatile int rcvPktFlag = 0;
volatile int rotateFlag = 0;
volatile int fwdFlag = 0;
const unsigned int angle[18] = {
63, 68, 74, 80, 85, 90, 95, 100, 104, 109, 113, 117, 121, 124, 127, 130, 133, 136,
};
const int forward[40] = {
99, 113, 131, 155, 183, 216, 253, 296, 354, 394,
435, 477, 521, 565, 610, 657, 704, 752, 802, 853,
904, 957, 1011, 1065, 1121, 1178, 1236, 1295, 1355,
1416, 1478, 1541, 1606, 1671, 1737, 1805, 1873, 1942, 2013, 2084
};

// Function Prototypes
void USART_Init();
void send_pkt(char msg);

int main(void)
{
    // Initialise USART
    USART_Init();

    int x;

    unsigned int sequence = 255;

    // Delay to allow for association with network
    _delay_ms(5000);

    while (1)
    {
        if (rcvpktFlag == 1) // If a packet has been received
            if (rxBuf[3] == ACKBYTE) // If the packet is an ACK
                // Don't think I need to worry about this as XBee module
                // handles ACK's
                // Do something to send an ACK back
                // OSSemPost(ReadySem);
            else
                // _delay_ms(100); // Not sure if need this delay
                // _delay_ms(100); // Not sure if need this delay
                // Send ACK of message back to host
                // OSSemPost(ReadySem);

                if (rcvpktFlag == 1) // If it's a new packet
                    sequence = rxBuf[5]; // Make the old sequence number
                    // the current packets sequence number
                        rotateFlag = 1;
                    if (rxBuf[3] == FORWARD)
                        fwdFlag = 1;
rcvPktFlag = 0; //Rest rcvPktFlag

if (rotateFlag == 1)
{
  //Do stuff to roate the required amount
  for (x=0;x<angle[rxBuf[4]];x++)
  {
    if (rxBuf[3] == PIVOTLEFT)
    {
      PORTC = 0xCC;  //1100 1100
      _delay_ms(1);
      PORTC = 0x66;  //0110 0110
      _delay_ms(1);
      PORTC = 0x33;  //0011 0011
      _delay_ms(1);
      PORTC = 0x99;  //1001 1001
      _delay_ms(1);
    }
    else if (rxBuf[3] == PIVOTRIGHT)
    {
      //First 4 digits are the motor on the switch side
      PORTC = 0xCC;  //1100 1100
      _delay_ms(1);
      PORTC = 0x99;  //1001 1001
      _delay_ms(1);
      PORTC = 0x33;  //0011 0011
      _delay_ms(1);
      PORTC = 0x66;  //0110 0110
      _delay_ms(1);
    }
    if (rotateFlag == 0) //If detect that rotate flag has been
turned off then break
    break;
  }
  if (rotateFlag == 1) //If E-Bug has finished movement without
being interrupted by the arrival of a new command
  {send_pkt(RDYBYTE);  //Signal host that movement is complete
  and E-Bug is ready for a new command
  }
  rotateFlag = 0; //Reset rotate flag
}

else if (fwdFlag == 1)
{
  //Do stuff to drive forward the required ammount
  //Drive forward
  for (x=0;x<forward[rxBuf[4]];x++)
  {
    if (rxBuf[3] == FORWARD)
    {
      PORTC = 0xCC;  //1100 1100
      _delay_ms(1);
      PORTC = 0x96;  //1001 0110
      _delay_ms(1);
    }
  }
}
_delay_ms(1);
PORTC = 0x33;  //0011 0011
_delay_ms(1);
PORTC = 0x69;  //0110 1001
_delay_ms(1);

if(fwdFlag == 0)  //If detect that forward flag has been
turned off the break
    break;

if(fwdFlag == 1)  //If E-Bug has finished movement without being
interrupted by the arrival of a new command
{
    send_pkt(RDYBYTE);  //Signal host that movement is complete
    //and E-Bug is ready for a new command
    fwdFlag = 0;  //Reset forward flag
}

return 0;

void USART_Init()
{
    //Set baud rate
    UBRR1L=UBRRL;  //low byte
    UBRR1H=(UBRRL>>8);  //high byte

    //Set data frame format: asynchronous mode, no parity, 1 stop bit, 8 bit
    //size
    //UPM11 = Parity mode, UPM10 = Type of parity check
    //See page 226 of 2561 data sheet
    // 1<<UMSEL11, got output. 0<<UMSEL11, no output. Shouldn't be the case
    //as the first option sets the USART to a reserved setting and the second to
    //Asynchronous mode
    UCSR1C=(1<<UMSEL11)|(0<<UMSEL10)|(0<<UPM11)|(0<<UPM10);  
    (0<<USB1)|(0<<UCSZ12)|(1<<UCSZ11)|(1<<UCSZ10);

    //Enable Transmitter and Receiver and Interrupt on receive complete
    UCSR1B=(1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1);

    //enable global interrupts
    sei();
}

void send_pkt(char msg)
{
    unsigned int packet[6];
packet[0] = 0xAA;
packet[1] = 0xFF;
packet[2] = ADDRESS;
packet[3] = msg;
packet[4] = 0x00;
packet[5] = 0x00;

for(int i = 0; i<6; i++)
{
    //Wait for empty transmit buffer
    //Waits for the transmit buffer to be empty by checking the UDREn
    (Data Register Empty) flag
    while (!((UCSR1A&(1<<UDRE1)))){};
    UDR1=packet[i];
}

ISR(USART1_RX_vect) //USART_RXC1_vect
{
    rxBuf[rxBufPtr] = UDR1;

    if(rxBuf[rxBufPtr] == PREAMBLE) //If its the start of a packet
    {
        rxBuf[0] = PREAMBLE;
        rxBufPtr = 0; //Set rxBufPtr to zero
    }

    if(rxBufPtr >= BUFFSIZE)
    {
        rxBufPtr = 0;
    }
    else {
        rxBufPtr++;
    }

    if((rxBuf[0]==0xAA) && rxBufPtr > 5) // If we've received a whole packet's worth
    {
        //When a new packet is received, if it is meant for us, clear all
        movement flags
        if ((rxBuf[1] == ADDRESS) && (rxBuf[2] == 0xFF)) //If packet
        destination address is us and source address = coordinator
        {
            rxBuf[0] = 0; // Prevent processing
            this packet again
            rcvPktFlag = 1; // Notify the handler
            that a complete packet has been received

            //Clear movement flags
            fwdFlag = 0;
            rotateFlag = 0;
        }
    }
    else {
        //
rcvPktFlag = 0;
}
13.3 Control Software Source Code (Visual C#)

/*
Date: 28/9/2009
Author: David McKechnie (modified by Rory Paltridge)
Version: 3
Status: In Development
Description:
This code was written by David McKechnie in 2008 and provides the base
station control and GUI for the E-Bug system.
It has been modified by Rory Paltridge due to the changes in the E-Bug end
device and to include increased functionality

Modifying code to handle multiple bugs. Changes made to...
-bgWrkerGetCoords_DoWork
*/

using System;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using System.Threading;
using MLApp;
using System.Drawing.Drawing2D;

namespace eBugControl
{
    public partial class Form1 : Form
    {
        delegate void SetTextCallback();
        delegate void SetDebugCallback(string text);

        MLAppClass _MLAppClass = new MLAppClass(); // Matlab Interface

        const byte FORWARD = 0xA; // Constants for sending commands
        const byte PIVOTRIGHT = 0x6;
        const byte PIVOTLEFT = 0x9;
        const byte ACK = 0x10;
        const byte RDY = 0x11;

        public SerialPort _serialPort = new SerialPort();
        public Semaphore ack_semaphore = new Semaphore(0, 1);
        public Semaphore auto_semaphore = new Semaphore(0, 1);

        public robotBug[] bugs; // A home for all the bugs
}
```csharp
public byte[] recvBuffer = new byte[1024];
public int recvBufferReadIndex = 0;
public int recvBufferWriteIndex = 0;

public Semaphore queue_sem = new Semaphore(1, 1);

public ArrayList lookup_color = new ArrayList(); //Lookup table for colour

public Form1()
{
    InitializeComponent(); //Set up and position all the buttons and stuff
}

// Runs at form startup
private void Form1_Load(object sender, EventArgs e)
{
    foreach (string s in SerialPort.GetPortNames())
    {
        listComPorts.Items.Add(s); // Populate the com port list
    }
    listComPorts.SelectedIndex = -1; // Select the first item in the lists
    listBauds.SelectedIndex = 0;
}

// When a user clicks the 'Connect' button
private void buttonComConnect_Click(object sender, EventArgs e)
{
    if (_serialPort.IsOpen) // Close any previously opened ports
    {
        comPortDebug.Text += "Closing " + _serialPort.PortName + "\n\n";
        _serialPort.Close();
    }
    _serialPort.PortName = listComPorts.Text; // Set the COM port
    _serialPort.BaudRate = int.Parse(listBauds.Text); // And the baud rate
    _serialPort.Parity = Parity.None;
    _serialPort.DataBits = 8;
    _serialPort.StopBits = StopBits.One;
    _serialPort.Handshake = Handshake.None;
    comPortDebug.Text += "Opening " + _serialPort.PortName + " @ " + _serialPort.BaudRate + "bps\n\n";
    try
    {
        _serialPort.Open();
    }
}
catch (IOException) // Com Port made no sense
{
  comPortDebug.Text += "ERR: That COM port doesn't exist!\r\n\r\n";
}
catch (InvalidOperationException) // Somehow, we have already opened it :
{
  comPortDebug.Text += "ERR: Com Port already open\r\n\r\n";
}
catch (UnauthorizedAccessException) // Some other program already opened it
{
  comPortDebug.Text += "ERR: Com Port in use, access denied\r\n\r\n";
}
if (_serialPort.IsOpen) // Everything worked!
{
  comPortDebug.Text += "Open!\r\n\n";
  if (panelCamControls.Enabled) panelRun.Enabled = true;
  // Enable the Run panel
  else panelRun.Enabled = false;
}

//Serial port data handler
void _serialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
  robotBug recvBug = new robotBug();
  //Temporary robotBug for receiving
  recvBug.packet = new byte[6];
  recvBug._serialPort = _serialPort;

  if (recvBufferWriteIndex + _serialPort.BytesToRead > recvBuffer.Length)
  {
    recvBufferWriteIndex = 0;
    //Reset the buffer if it got full
    recvBufferReadIndex = 0;
  }
  if (_serialPort.BytesToRead > recvBuffer.Length)
    _serialPort.DiscardInBuffer();
  //Discard it if there's too much
  recvBufferWriteIndex += _serialPort.Read(recvBuffer, recvBufferWriteIndex, _serialPort.BytesToRead);
  for (; recvBufferReadIndex < recvBufferWriteIndex; recvBufferReadIndex++)
  {
    if (recvBuffer[recvBufferReadIndex] == 0xAA)
      //If its the start of a packet
    {
      //Serial port data handler
      void _serialPort_DataReceived
Array.Copy(recvBuffer, recvBufferReadIndex,
    //Copy it to the start of the buffer
    recvBuffer, 0,
    recvBuffer.Length - recvBufferReadIndex);
recvBufferWriteIndex -= recvBufferReadIndex;

    //Adjust the indexes
recvBufferReadIndex = 0;
    }
    if ((recvBuffer[0] == 0xAA) && recvBufferWriteIndex > 5)
        //If we have a whole packet
    {
        recvBuffer[0] = 0;
        Array.Copy(recvBuffer, recvBug.packet, 6);

        //Copy it into the receivebug
        recvBug.recvPacket();
        if (recvBug.dstaddress == 0xFF)
            //Don't have to worry about this as XBee handles
            corrupt packets and ACK's itself
                //if (recvBug.CRC8() == recvBug.packet[6])
            //If the packet is not corrupt
                //
                // if (recvBug.direction == ACK)
            //If its an ACK, tell whoever
            // try { ack_semaphore.Release(); }            //sent
            // catch (SemaphoreFullException) { }
            if (recvBug.direction == RDY)
                //If
                //set
                the appropriate bug to RDY
                if (bugs[recvBug.srcaddress].ready != true)
                    {
                        bugs[recvBug.srcaddress].ready = true;
                        try { auto_semaphore.Release(); } //tell the queue that someone
                        catch (SemaphoreFullException) { }    //is
                        ready
                        updateBugStatus();
                        }
                // recvBug.dstaddress = recvBug.srcaddress;
                // recvBug.srcaddress = 0xFF;
                // recvBug.direction = ACK;
                // recvBug.length_index = 0x0;
                // recvBug.sequence = 0x0;
                // recvBug.sendCommandNoRepeat();
            //Send an ACK back to the RDY
            }
        //}
        }
    }

private void trackBarDistance_Scroll(object sender, EventArgs e)
int value = (trackBarDistance.Value / 5) * 5;
labelDistanceDisp.Text = value.ToString();
}

private void trackBarAngle_Scroll(object sender, EventArgs e)
{
    int value = (trackBarAngle.Value / 10) * 10;
    labelAngleDisp.Text = value.ToString();
}

// When a user clicks the 'Set' bugs button
private void buttonSetBugs_Click(object sender, EventArgs e)
{
    bugs = new robotBug[(int)numBugs.Value];

    // Setup colour lookup table
    lookup_color.Add("Red");
    lookup_color.Add("Green");
    lookup_color.Add("Blue");

    comboBugSelect.Items.Clear();
    for (byte i = 0; i < numBugs.Value; i++) // Create a bunch of bugs with default values
    {
        bugs[i] = new robotBug();
        bugs[i]._Form = this;
        bugs[i].packet = new byte[6];
        bugs[i].ready = true;
        bugs[i].x = 0;
        bugs[i].y = 0;
        bugs[i].radius = 0;
        bugs[i].angle = 0;
        bugs[i].color = i; // Arbitrarily assign each bug a colour 0: Red, 1: Green, 2: Blue
        bugs[i].srcaddress = 0xFF;
        bugs[i].dstaddress = i; // Arbitrarily assign each bug an address
        bugs[i].sequence = 0x00;
        bugs[i]._serialPort = _serialPort; // Point them all at the same serial port
        bugs[i].acksem = ack_semaphore;
        bugs[i].auto_stage = 0;
        comboBugSelect.Items.Add("Bug " + bugs[i].dstaddress);
        comboAutoBugSelect.Items.Add("Bug " + bugs[i].dstaddress);
        // Add the data received event handler
        _serialPort.DataReceived += new
        SerialDataReceivedEventHandler(_serialPort_DataReceived);
    }

    // Display some bug stats
    comboBugSelect.SelectedIndex = 0;
    comboAutoBugSelect.SelectedIndex = 0;
groupManual.Enabled = true;
int[] CalValues = new int[2];
CalValues[0] = trackBarBlackCal.Value;
bgWrkerGetCoords.RunWorkerAsync(CalValues);  //Start running the
thread to grab data from Matlab
bgWrkerAutoControl.RunWorkerAsync();  //Start the queuing
thread
}

//Called whenever a bug changes its status
public void updateBugStatus()
{
    if (this.textboxBugStatus.InvokeRequired)  //If it wasn't
called by the right thread
    {
        SetTextCallback d = new SetTextCallback(updateBugStatus);
        this.Invoke(d, new object[] { new object });  //Tell the right
thread that there's work to do
    }
    else  //It was called
    {
        textboxBugStatus.Clear();
textboxBugStatus.Text += "\tRDY?\tX\tY\tR\tAngle\n";
        foreach (robotBug _robotBug in bugs)
        {
            textboxBugStatus.Text += "Bug " + _robotBug.dstaddress + "\t"
if (_robotBug.ready) textboxBugStatus.Text += "RDY"
            textboxBugStatus.Text += "\t" + _robotBug.x.ToString("000.0")
+ "\t" + _robotBug.y.ToString("000.0")
+ "\t" + _robotBug.radius.ToString("00")
+ "\t" + _robotBug.angle.ToString("0.00")
textboxBugStatus.Text += "\r\n";
            panelBugSpace.Invalidate();  //Invalidate
            the drawspace, so it gets repainted
        }
    }
}

public void writeRunDebug(string text)
{
    if (this.textRunDebug.InvokeRequired)
    {
        SetDebugCallback d = new SetDebugCallback(writeRunDebug);
        this.Invoke(d, new object[] { text });
    }
    else
        textRunDebug.Text += text;
}

//Manual control buttons
private void buttonForward_Click(object sender, EventArgs e) {
    int bug_index = int.Parse(comboBugSelect.Text.Substring(4, 1));
    bugs[bug_index].direction = FORWARD;
    bugs[bug_index].length_index = (byte)(byte.Parse(labelDistanceDisp.Text) / 5 - 1);
    bugs[bug_index].sendCommand();
    updateBugStatus();
}

private void buttonLeft_Click(object sender, EventArgs e) {
    int bug_index = int.Parse(comboBugSelect.Text.Substring(4, 1));
    bugs[bug_index].direction = PIVOTLEFT;
    bugs[bug_index].length_index = (byte)(byte.Parse(labelAngleDisp.Text) / 10 - 1);
    bugs[bug_index].sendCommand();
    updateBugStatus();
}

private void buttonRight_Click(object sender, EventArgs e) {
    int bug_index = int.Parse(comboBugSelect.Text.Substring(4, 1));
    bugs[bug_index].direction = PIVOTRIGHT;
    bugs[bug_index].length_index = (byte)(byte.Parse(labelAngleDisp.Text) / 10 - 1);
    bugs[bug_index].sendCommand();
    updateBugStatus();
}

private void buttonFindPath_Click(object sender, EventArgs e) {
    if (folderBrowserMATLAB.ShowDialog() == DialogResult.OK) //Some wonderful .Net magic
    {
        textBoxPath.Text = folderBrowserMATLAB.SelectedPath;
    }
}

private void buttonOpen_Click(object sender, EventArgs e) {
    bool err = false;
    _MLAppClass.Execute("closepreview"); //Get matlab to close any old previews
    _MLAppClass.Execute("vid = videoinput('winvideo',1,'RGB24_320x240');"); //Open the webcam YUY2_320x240
    //See videoinput in MATLAB
    obj = videoinput(adaptorname,deviceID,format,P1,V1,...) creates a video input object obj with the specified property values. If an invalid property name or property value is specified, the object is not created.
    */
    try
    {
        object matlabOut = _MLAppClass.GetVariable("vid", "base");
catch (System.Runtime.InteropServices.COMException) //Generic error, we'll assume the camera is wonky
{
    matlabDebug.Text += "Eek! Error opening video object... (is the camera plugged in?)\n\n";
    panelCamControls.Enabled = false;
    err = true;
}
if (!err)
{
    matlabDebug.Text += "Video object open\n";
    _MLAppClass.Execute("src_vid = getselectedsource(vid);");
    //Lets us change things about the camera
    _MLAppClass.Execute("triggerconfig(vid, 'manual');");
    //So it doesn't take an eternity to grab frames
    _MLAppClass.Execute("set(src_vid, 'Focusmode', 'manual');");
    // Stops the dreaded autofocus
    _MLAppClass.Execute("cd '" + textBoxPath.Text + "';");
    matlabDebug.Text += "Set path to '" + textBoxPath.Text + "'\n";
    _MLAppClass.Execute("preview(vid);");
    //Open a preview box
    matlabDebug.Text += "Video Preview Started\n";
    panelCamControls.Enabled = true;
    if (_serialPort.IsOpen) panelRun.Enabled = true;
    //Enable the run panel if the comport stuff is already done
}
}
private void matlabDebug_TextChanged(object sender, EventArgs e)
{
    matlabDebug.SelectionLength = comPortDebug.Text.Length;
    matlabDebug.ScrollToCaret();
}
private void comPortDebug_TextChanged(object sender, EventArgs e)
{
    comPortDebug.SelectionLength = comPortDebug.Text.Length;
    comPortDebug.ScrollToCaret();
}
private void trackBarPan_Scroll(object sender, EventArgs e)
{
    _MLAppClass.Execute("set(src_vid, 'Pan', " + trackBarPan.Value + ")");
}
private void trackBarZoom_Scroll(object sender, EventArgs e)
{
    _MLAppClass.Execute("set(src_vid, 'Zoom', " + trackBarZoom.Value + ")");
}
private void trackBarTilt_Scroll(object sender, EventArgs e)
{
    _MLAppClass.Execute("set(src_vid, 'Tilt', " + trackBarTilt.Value + ")");
}

//Display the calibration windows when necessary
private void trackBarWhiteCal_MouseUp(object sender, MouseEventArgs e)
{
    _MLAppClass.Execute("test = getsnapshot(vid)";
    _MLAppClass.Execute("figure(2)");
    _MLAppClass.Execute("imshow(im2bw(test," + trackBarWhiteCal.Value + ")/100))");
}

private void trackBarBlackCal_MouseUp(object sender, MouseEventArgs e)
{
    _MLAppClass.Execute("test = getsnapshot(vid)";
    _MLAppClass.Execute("figure(3)");
    _MLAppClass.Execute("imshow(im2bw(test," + trackBarBlackCal.Value + ")/100))");
}

//Gets the bug locations from the MATLAB environment
private void bgWrkerGetCoords_DoWork(object sender, DoWorkEventArgs e)
{
    Array XReal = new double[5];
    Array XImag = new double[5];

    int[] CalValues = (int[])e.Argument;

double count, color;
int i;
int ident_err_ctr = 0; //Number of colour identification errors

    _MLAppClass.Execute("start(vid);");
    //Starts the
for (; ; )
{
    if (bgWrkerGetCoords.CancellationPending)
        break;
    _MLAppClass.Execute("bugs = bugfinder3(getsnapshot(vid)," +
//bugfinder3(ycbcr2rgb(getsnapshot(vid)))
        CalValues[0] + "/100," + //Capture and
    process and image
        CalValues[1] + "/100, 20);"));
    _MLAppClass.Execute("[m n] = size(bugs);");
    try
    {
        count = (double)_MLAppClass.GetVariable("m", "base");
        //Check if there were any results
catch (System.Runtime.InteropServices.COMException)
{
    continue;
}
if (count != 0) //There were results!
{
    for (i = 0; i < count; i++) //Count through the number of bugs found
    {
        string command = "temp = bugs(" + i.ToString() + "+1),:);"; //Concatinates the value of i into the MATLAB command string so it now reads "temp = bugs((i+1),:);"
        _MLAppClass.Execute(command); //Executes MATLAB command which is written in string on previous line
        _MLAppClass.GetFullMatrix("temp", "base", ref XReal, ref XImag); //Get the values from MATLAB

        color = (double)XReal.GetValue(4); //The colour of the identified bug

        if (color == -1) //If the colour couldn't be determined by MATLAB
        {
            //Do nothing, hopefully it will be identified correctly next time
            ident_err_ctr++; //Increment the identification error counter
        }
        else
        {
            //Assign all the values from MATLAB to the relevantly coloured bug
            bugs[(int)color].y = (double)XReal.GetValue(0);
            bugs[(int)color].x = (double)XReal.GetValue(1);
            bugs[(int)color].radius = (double)XReal.GetValue(2);
            bugs[(int)color].angle = (double)XReal.GetValue(3);
            updateBugStatus();
        }
    }
    //Thread.Sleep(0);                  //Probably better to include this, but it makes things slow :(
}

private void trackBarXDest_Scroll(object sender, EventArgs e)
{
    labelXDestDisp.Text = trackBarXDest.Value.ToString();
}

private void trackBarYDest_Scroll(object sender, EventArgs e)
{
    labelYDestDisp.Text = trackBarYDest.Value.ToString();
}
What to do when the BugSpace panel needs to be redrawn

```csharp
private void panelBugSpace_Paint(object sender, PaintEventArgs e)
{
    Point offset = new Point(20, 20); // An offset, to stop us drawing _right_ in the corner

    Graphics _Graphics = e.Graphics; // Use the form graphics object we were passed

    if (bugs != null) // If the bugs array has been populated
    {
        foreach (robotBug _robotBug in bugs)
        {
            if (_robotBug.radius != 0) // If this seems to be a real bug
            {
                Rectangle Circle = new Rectangle( // ahahah Rectangle circle. It's a bounding box
                    Convert.ToInt32(offset.X + _robotBug.x - _robotBug.radius),
                    Convert.ToInt32(offset.Y + _robotBug.y - _robotBug.radius),
                    Convert.ToInt32(_robotBug.radius * 2),
                    Convert.ToInt32(_robotBug.radius * 2));
                switch ((int)_robotBug.color) // Depending upon the colour of the bug we're looking at...
                {
                    case 1: // Green E-Bug
                        LinearGradientBrush linearBrushGreen = new LinearGradientBrush(Circle,
                        Color.Green, Color.DarkGreen,
                        LinearGradientMode.ForwardDiagonal);
                        _Graphics.FillEllipse(linearBrushGreen, Circle); // Fill a green ellipse in the circle box
                        break;
                    case 2: // Blue E-Bug
                        LinearGradientBrush linearBrushBlue = new LinearGradientBrush(Circle,
                        Color.Blue, Color.DarkBlue,
                        LinearGradientMode.ForwardDiagonal);
                        _Graphics.FillEllipse(linearBrushBlue, Circle); // Fill a blue ellipse in the circle box
                        break;
                    default: // Red E-Bug
                        LinearGradientBrush linearBrushRed = new LinearGradientBrush(Circle,
                        Color.Red, Color.DarkRed, // DarkOrange
                        LinearGradientMode.ForwardDiagonal);
                        _Graphics.FillEllipse(linearBrushRed, Circle); // Fill a red ellipse in the circle box
                        break;
                }
            }
        }
    }
}
```
Circle = new Rectangle( //Create a new circle inside the previous circle, 70% the size
    _robotBug.radius * 0.7),
    Convert.ToInt32(offset.X + _robotBug.x -
    _robotBug.radius * 0.7),
    Convert.ToInt32(offset.Y + _robotBug.y -
    _robotBug.radius * 0.7),
    Convert.ToInt32(_robotBug.radius * 1.4),
    Convert.ToInt32(_robotBug.radius * 1.4));

SolidBrush solidBrush = new SolidBrush(Color.Black);
//Fill a black ellipse over the coloured one
Pen pen = new Pen(Color.White,
    Convert.ToInt32(_robotBug.radius * 0.2));
_Graphics.FillEllipse(solidBrush, Circle);
//Draw the white angle
line
    Convert.ToInt32(offset.X + _robotBug.x),
    Convert.ToInt32(offset.Y + _robotBug.y),
    Convert.ToInt32(_robotBug.x + _robotBug.y +
    _robotBug.radius * Math.Cos(_robotBug.angle)),
    Convert.ToInt32(_robotBug.y +
    _robotBug.radius * Math.Sin(_robotBug.angle)));
pen = new Pen(Color.Green, 1); //Draw the crosshairs
_Graphics.DrawLine(pen, trackBarXDest.Value + 20, 20,
trackBarXDest.Value + 20, 260);
_Graphics.DrawLine(pen, 20, trackBarYDest.Value + 20,
340, trackBarYDest.Value + 20);

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    bgWrkerGetCoords.CancelAsync(); //Try to cleanly exit the background worker
    _MLAppClass.Quit();
}

private void bgWrkerAutoControl_DoWork(object sender, DoWorkEventArgs e)
{
    int[] calc_vals;

    for (; ; )
    {
        auto_semaphore.WaitOne(); //Wait for someone to say they're ready for more tasks
        foreach (robotBug _robotbug in bugs)
        {
            //Autocontrol function

            foreach (robotBug _robotbug in bugs)
if ( (_robotbug.queue.Count != 0) && (_robotbug.ready == true) ) // If there are queued instructions that's ready
{
    queue_sem.WaitOne();
    calc_vals = angler(); // Work out how to get to the next location

    if (_robotbug.ready == true) // Find the bug
    {
        queue_sem.WaitOne();
        calc_vals = angler(); // Work out how to get

        // Where you want to go minus where you are - x
        (int)_robotbug.queue[0] - Convert.ToInt32(_robotbug.x),

        // Where you want to go minus where you are - y
        (int)_robotbug.queue[1] - Convert.ToInt32(_robotbug.y),

        // Current angle of the robot
        _robotbug.angle);

        queue_sem.Release();

        if (!_robotbug.move_state) // Turn first
        {
            if (Math.Abs(calc_vals[1]) >= 10) // If it's a big enough angle to worry about
            {
                _robotbug.length_index = (byte)(Math.Abs(calc_vals[1]) / 10 - 1); // Calculate the angle to turn
                if (calc_vals[1] < 0)
                {
                    _robotbug.direction = PIVOTLEFT;
                }
                else
                {
                    _robotbug.direction = PIVOTRIGHT;
                }
                _robotbug.sendCommand(); // Send pivot command to E-Bug
            }
        }
        else // Move second
        {
            if (calc_vals[0] != 0) // If we aren't already at our destination
            {
                _robotbug.length_index = Convert.ToByte((calc_vals[0] * (5 / _robotbug.radius)) / 5 - 1); // Calculate the length to be sent to the E-Bug
                _robotbug.direction = FORWARD; // Set E-Bug direction
                // Thread.Sleep(100); // Don't think I need this
                _robotbug.sendCommand(); // Send forward command to E-Bug
            }
        } // End if auto stage
        _robotbug.auto_queue.GetLength(0) /*if (_robotbug.auto_stage >=
        _robotbug.auto_queue.GetLength(0))
        {
            if (_robotbug.repeat)
            {
                _robotbug.auto_stage = 0;
                else
                {
                    _robotbug.auto_queue = null;
                    _robotbug.queue.Clear();
                } */
    } // End if ready
Real Time Control Over Wireless Links

}`
_robotbug.move_state = !_robotbug.move_state; //Flip
from turn to move, and vice versa
}

if (_robotbug.repeat) //If repeat is true...
{
  //Add just executed command back on to end of queue
  _robotbug.queue.Add(_robotbug.queue[0]); //X
  coordinate

  _robotbug.queue.Add(_robotbug.queue[1]); //Y
  coordinate
}

_robotbug.queue.RemoveRange(0, 1); //Delete executed
command from queue

}  

private int[] angler(int x, int y, double angle)
{
  int[] calc_vals = new int[2];
  calc_vals[0] = Convert.ToInt32(Math.Sqrt(Math.Pow(x, 2) +
Math.Pow(y, 2))); //Work out how far away the destination is

  if ((y == 0) || (x == 0)) //If y == 0 || y == 0
  {  //This line
    calc_vals[1] = 0; //We need to go in a straight line
  }
  else
  {
    calc_vals[1] = Convert.ToInt32(180 *
Math.Abs(Math.Atan(Convert.ToDouble(y) / Convert.ToDouble(x)))) / Math.PI; //Find the angle

    if (x <= 0) //Find the quadrant
      if (y <= 0)
        calc_vals[1] = -180 + (calc_vals[1]);
      else
        calc_vals[1] = 180 - (calc_vals[1]);
    else
      if (y < 0)
        calc_vals[1] = -calc_vals[1];
      else
        calc_vals[1] = calc_vals[1];

    calc_vals[1] -= Convert.ToInt32(angle * 180 / Math.PI);

    if (calc_vals[1] > 180)
      calc_vals[1] -= 360; //Turning a long way left is the same
    as turning a little bit right :D
    if (calc_vals[1] < -180)
      calc_vals[1] += 360;

    return calc_vals;
  }

private void buttonAutoGo_Click(object sender, EventArgs e)
{
    int bug_index = int.Parse(comboAutoBugSelect.Text.Substring(4, 1));
    queue_sem.WaitOne();
    //bugs[bug_index].auto_queue = new int[1, 2]; // new int[1, 2] is 1x2 dimensional array declaration
    //Add an item to the queue
    //bugs[bug_index].auto_queue[0, 0] = trackBarXDest.Value;
    bugs[bug_index].queue.Add(trackBarXDest.Value);
    //bugs[bug_index].auto_queue[0, 1] = trackBarYDest.Value;
    bugs[bug_index].queue.Add(trackBarYDest.Value);
    bugs[bug_index].repeat = false;
    //bugs[bug_index].auto_stage = 0;
    bugs[bug_index].move_state = false;
    bugs[bug_index].ready = true;
    queue_sem.Release();
    try { auto_semaphore.Release(); } //Tell the queuing system we made a change
    catch (SemaphoreFullException) { }
}
//Repeat a square shape indefinitely
private void buttonSquare_Click(object sender, EventArgs e)
{
    int bug_index = int.Parse(comboAutoBugSelect.Text.Substring(4, 1));
    queue_sem.WaitOne();
    //First corner of square
    bugs[bug_index].queue.Add(100); //x
    bugs[bug_index].queue.Add(100); //y
    //Second corner of square
    bugs[bug_index].queue.Add(100); //x
    bugs[bug_index].queue.Add(200); //y
    //Third corner of square
    bugs[bug_index].queue.Add(200); //x
    bugs[bug_index].queue.Add(200); //y
    //Fourth corner of square
    bugs[bug_index].queue.Add(200); //x
    bugs[bug_index].queue.Add(100); //y
    bugs[bug_index].repeat = true;
    bugs[bug_index].ready = true;
    //bugs[bug_index].auto_stage = 0;
    bugs[bug_index].move_state = false;
    queue_sem.Release();
    try { auto_semaphore.Release(); } //Tell the queue we changed something
    catch (SemaphoreFullException) { }
}
/Quit Program
private void Quit_Click(object sender, EventArgs e)
{
    bgWrkerGetCoords.CancelAsync(); //Try to cleanly exit the
    background worker
    bgWrkerAutoControl.CancelAsync();
    _MLAppClass.Quit();
    Application.Exit();
}

public class robotBug
{
    public Form1 _Form;
    public bool ready;
    public SerialPort _serialPort;
    public Semaphore acksem;
    public byte sequence;
    public byte srcaddress;
    public byte dstaddress;
    public byte direction;
    public byte length_index;
    public byte[] packet;
    //public int auto_stage;
    public bool move_state;
    //public int[,] auto_queue;
    public ArrayList queue = new ArrayList();
    public bool repeat;
    public double x;
    public double y;
    public double angle;
    public double radius;
    public double color;

    public void sendCommand() //Complete sending operation, including
    auto retries
    {
        Random rand = new Random();
        //bool _continue = true;
        //int repeat = 0;
        packet[0] = 0xAA;
        packet[1] = dstaddress;
        packet[2] = srcaddress;
        packet[3] = direction;
        packet[4] = length_index;
        packet[5] = sequence++;
        //packet[6] = CRC8();
        _serialPort.Write(packet, 0, packet.Length);
/* while (_continue)
{ 
    for (int i = 0; i < 5; i++)
    {
        _serialPort.Write(packet, 0, 1);
    }
    _serialPort.Write(packet, 0, packet.Length);

    _continue = !acksem.WaitOne(rand.Next(50, 150), false);
    ready = _continue;
    if (++repeat > 100)
    {
        _Form.writeRunDebug("Timed out sending to " + dstaddress + " :( \r\n");
        //_Form.textRunDebug.AppendText("Timed out sending to " + dstaddress + " :( \r\n");
        _continue = false;
    }
} */

public void sendCommandNoRepeat()    //Used for one off sends (like an ACK)
{
    packet[0] = 0xAA;
    packet[1] = dstaddress;
    packet[2] = srcaddress;
    packet[3] = direction;
    packet[4] = length_index;
    packet[5] = sequence++;
    //packet[6] = CRC8();
    _serialPort.Write(packet, 0, packet.Length);
}

/*
public byte CRC8()
{
    byte CRC = 0xFF;
    for (int i = 1; i < (packet.Length - 1); i++)
    {
        CRC = (byte)(CRC ^ packet[i]);
        for (int j = 0; j < 8; j++)
        {
            CRC = ((CRC & 0x80) != 0) ? (byte)((CRC << 1) ^ 0x1D) : (byte)(CRC << 1);
        }
    }
    return CRC;
} */

public void recvPacket()    //Just populate the members from the packet
{
    dstaddress = packet[1];
    srcaddress = packet[2];
    direction = packet[3];
length_index = packet[4];
sequence = packet[5];
}
13.4 MATLAB Source Code

function circles = bugfinder3(img_in,blackthres,whitethres,gridsize)

    circles = [];
    img_black = im2bw(img_in, blackthres);  % Black calibrated image
    img_white = im2bw(img_in, whitethres);   % White calibrated image

    %********** Low res image search ******************
    % This section downsamples the black image by taking every 20th pixel in x and y directions
    % Ideally each bug appears as only a single white pixel. This downsized image is searched to identify white pixels (the bugs)
    % The function limitfinder() is then called to find the borders of the detected bug, on the full res image, to verify it is actually a bug
    % If it is a bug, not an artifact, then findangle() is called to find y0, x0, radius and angle of the bug
    % A black pixel is a 0, a white pixel a 1

    grid1 = 1:gridsize:size(img_in,1);
    grid2 = 1:gridsize:size(img_in,2);

    img_black_small = img_black(grid1,grid2);

    index = 1;
    for j = 2:1:size(img_black_small,2)-1
        for i = 2:1:size(img_black_small,1)-1
            if(img_black_small(i,j)) % If a white pixel is detected
                limits = limitfinder(img_white, i, j, gridsize, img_in);  % Call the limitfinder function to make sure it is an actual bug...
                bug_colour = findbugcolour(pix_colour); % Function returns the colour of the bug that has been found. Used to identify distinguish individual bugs
                if(~limits(4)) % if limitfinder didn't fail, find the angle
                    angle = findangle(limits(1),limits(2),limits(3),img_black);  % Call the findangle function to find x0, y0, radius and angle of the detected bug
                    if numel(angle)~=0;  %if the number of elements in angle is != to zero
                        circles(index,:) = [limits(1:3) angle bug_colour];
                        %store y0, x0, radius and angle to 'index' element of circles
                        index = index + 1;
                        img_black_small(i-1:i+1,j-1:j+1) = 0;
                    end
                end
            end
        end
    end

%******************************************************************************

function [limits, pix_colour] = limitfinder(img_white, i, j, gridsize, img_in)
i_big = (i-1)*gridsize+1;
j_big = (j-1)*gridsize+1;
n = i_big;
detectstate = 0;
umblack = 0;
fail = 0;
%First find the bottom of the circle
while(numblack < 1)
    if(img_white(n,j_big))
        % Found another white pixel
        if(detectstate)
            numblack = 0;
detectstate = 0;
        end
    else
        %Found a black pixel
        detectstate = 1;
umblack = numblack + 1;
pix_colour(1,:) = img_in(n,j_big,1:3); % Save the RGB values of the pixel at this point
        end
end
n = n+1;
if((abs(n-i_big)>50)||n>size(img_white,1)) %If this executes we have failed to identify a bug
    fail = 1; %Failed to identify a bug
    break;
end
bottom = n;

%Then find the top of the circle
n = i_big;
detectstate = 0;
umblack = 0;
while(numblack < 1)
    if(img_white(n,j_big))
        % Found another white pixel
        if(detectstate)
            numblack = 0;
detectstate = 0;
        end
    else
        %Found a black pixel
        detectstate = 1;
umblack = numblack + 1;
pix_colour(2,:) = img_in(n,j_big,1:3); % Save the RGB values of the pixel at this point
        end
end
n = n-1;
if((abs(n-i_big)>50)||n<1))
    fail = 1;
    break;
end
top = n;
%Find the right limit
n = j_big;
detectstate = 0;
umblack = 0;
while (numblack < 1)
  if (img_white(i_big,n))
    % Found another white pixel
    if (detectstate)
      numblack = 0;
detectstate = 0;
  end
  else
    %Found a black pixel
    detectstate = 1;
    numblack = numblack + 1;
pix_colours(:, :) = img_in(i_big,n,1:3); % Save the RGB values of the
    pixel at this point
  end
  n = n+1;
  if((abs(n-j_big)>50) || (n>size(img_white,2)))
    fail = 1;
    break;
  end
end
right = n;
if (~fail)
  M = [top^2+j_big^2, top, j_big, 1;
bottom^2+j_big^2, bottom, j_big, 1;
i_big^2+right^2, i_big, right, 1];

  m_11 = det(M(:,[2 3 4]));
  m_12 = det(M(:,[1 3 4]));
  m_13 = det(M(:,[1 2 4]));
  m_14 = det(M(:,[1 2 3]));

  y = 0.5*m_12/m_11;
x = -0.5*m_13/m_11;
r = sqrt(y^2 + x^2 + m_14/m_11);
  limits = [y x r 0];
else
  limits = [0 0 0 1];
end

function angle = findangle(y0, x0, r, img_black)

  r = round(0.6*r);
x0 = round(x0);
y0 = round(y0);

  angle = [];
\[
\text{if}((x_0-r>0) \&\& (x_0+r<\text{size(img_black,2)}) \&\& (y_0-r>0) \&\& (y_0+r<\text{size(img_black,1)}))
\]

\[
n = 1;
\]

\[
f = 1-r;
\]

\[
\text{ddF}_x = 0;
\]

\[
\text{ddF}_y = 0;
\]

\[
x = 0;
\]

\[
y = r;
\]

\[
\text{if}(\neg \text{img_black}(y_0+r, x_0))
\]

\[
\quad \text{blackpix}(n,:) = [y_0+r, x_0];
\]

\[
\quad n = n + 1;
\]

\[
\text{if}(\neg \text{img_black}(y_0-r, x_0))
\]

\[
\quad \text{blackpix}(n,:) = [y_0-r, x_0];
\]

\[
\quad n = n + 1;
\]

\[
\text{if}(\neg \text{img_black}(y_0, x_0+r))
\]

\[
\quad \text{blackpix}(n,:) = [y_0, x_0+r];
\]

\[
\quad n = n + 1;
\]

\[
\text{if}(\neg \text{img_black}(y_0, x_0-r))
\]

\[
\quad \text{blackpix}(n,:) = [y_0, x_0-r];
\]

\[
\quad n = n + 1;
\]

\[
\text{end}
\]

\[
\text{while}(x < y)
\]

\[
\quad \text{if}(f >= 0)
\]

\[
\quad \quad y = y - 1;
\]

\[
\quad \quad \text{ddF}_y = \text{ddF}_y + 2;
\]

\[
\quad \quad f = f + \text{ddF}_y;
\]

\[
\quad x = x + 1;
\]

\[
\quad \text{ddF}_x = \text{ddF}_x + 2;
\]

\[
\quad f = f + \text{ddF}_x + 1;
\]

\[
\quad \text{if}(\neg \text{img_black}(y_0+y, x_0+x))
\]

\[
\quad \quad \text{blackpix}(n,:) = [y_0+y, x_0+x];
\]

\[
\quad \quad n = n + 1;
\]

\[
\text{if}(\neg \text{img_black}(y_0+y, x_0-x))
\]

\[
\quad \quad \text{blackpix}(n,:) = [y_0+y, x_0-x];
\]

\[
\quad \quad n = n + 1;
\]

\[
\text{if}(\neg \text{img_black}(y_0-y, x_0+x))
\]

\[
\quad \quad \text{blackpix}(n,:) = [y_0-y, x_0+x];
\]

\[
\quad \quad n = n + 1;
\]

\[
\text{if}(\neg \text{img_black}(y_0-y, x_0-x))
\]

\[
\quad \quad \text{blackpix}(n,:) = [y_0-y, x_0-x];
\]

\[
\quad \quad n = n + 1;
\]

\[
\text{end}
\]

\[
\text{if}(\neg \text{img_black}(y_0+x, x_0+y))
\]

\[
\quad \quad \text{blackpix}(n,:) = [y_0+x, x_0+y];
\]

\[
\quad \quad n = n + 1;
\]

\[
\text{end}
\]

\[
\text{end}
\]

\[
\text{end}
\]
if (~img_black(y0+x, x0-y))
    blackpix(n,:) = [y0+x, x0-y];
    n = n + 1;
end
if (~img_black(y0-x, x0+y))
    blackpix(n,:) = [y0-x, x0+y];
    n = n + 1;
end
if (~img_black(y0-x, x0-y))
    blackpix(n,:) = [y0-x, x0-y];
    n = n + 1;
end

if (n>1)
    dot = mean(blackpix);
elseif (n>1)
    dot = blackpix;
end
if (n==1)
    angle = atan2(dot(1)-y0, dot(2)-x0);
end
end
end

function bug_colour = findbugcolour(pix_colour)

% Red = 0
% Green = 1
% Blue = 2

%Calculate the RGB average values
rgb_avg = sum(pix_colour,1) / 3; %sum the rows and divide by 3
red_avg = rgb_avg(1);
green_avg = rgb_avg(2);
blue_avg = rgb_avg(3);

if( (red_avg > green_avg) && (red_avg > blue_avg) ) %If Red is the
    bug_colour = 0; %Red
elseif( (green_avg > red_avg) && (green_avg > blue_avg) ) %If Green is
    bug_colour = 1; %Green
elseif( (blue_avg > red_avg) && (blue_avg > green_avg) ) %If Blue is the
    bug_colour = 2; %Blue
else
    bug_colour = -1; %Error - Couldn't choose a colour
end
end
13.5 E-Bug Logic Circuit Schematic
13.6 E-Bug Logic Circuit PCB
13.7 E-Bug Power Circuit Schematic
13.8 E-Bug Power PCB