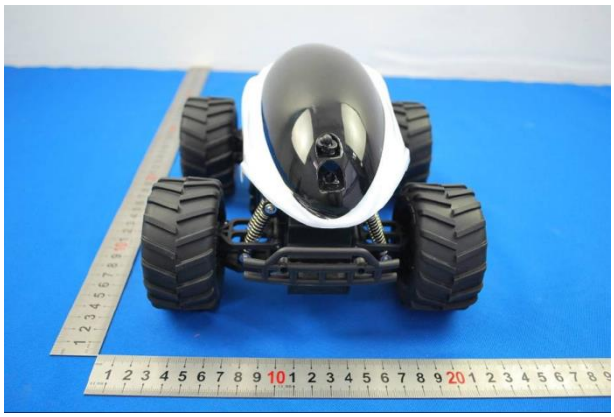
Python SDK for the BeeWi Camera Buggy Documentation

Summary of the Buggy

The Buggy[1] is an off road toy car fitted with a VGA video camera and a Wi-Fi hotspot. A free application[2] (compatible with iOS and Android) allows users to control the Buggy to up to roughly 11mph, whilst viewing the video feed in real time.



The Buggy takes 2 hours to fully charge and can be used for up to 20 minutes before needing to recharge.

They can be bought quite cheaply from various distributors[3], which make them great to use in schools and for learning how to program using this Standard Development Kit.

A YouTube video showing some of the functions of the SDK is available at https://www.youtube.com/watch?v=IKUlAkl-0oc .

[1] Bee-wi.com, (2014). BWZ200 - Wifi Camera Buggy. [online] Available at: http://www.bee-wi.com/wifi-camera-buggy-bwz200-a1-beewi,us,4,BWZ200-A1.cfm [Accessed 10 Sep. 2014].
[2] Bee-wi.com, (2014). BeeWi BuggyPad. [online] Available at: http://www.bee-wi.com/buggypad-application-beewi,us,4,BuggyPad-App.cfm [Accessed 10 Sep. 2014].
[3] Buggy, B. (2014). BeeWi WiFi Camera Scara Bee Buggy | Maplin. [online] Maplin.co.uk. Available at: http://www.maplin.co.uk/p/beewi-wifi-camera-scara-bee-buggy-n65qt [Accessed 16 Sep. 2014].

Getting Started

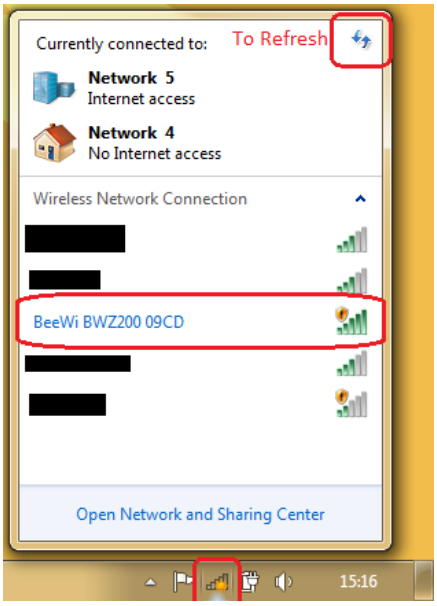
Connecting to the Buggy

Once you have got hold of a BeeWi Camera Scara Bee Buggy, the first thing to do is to connect to it via a laptop or PC.

To do this, first make sure your Buggy is charged (leave charging for 2 hours to fully charge). Then turn the Buggy on via the power switch located on the underside of the Buggy. When the Buggy is on, it will start emitting a Wi-Fi signal you can connect to.

Note: *The following steps are specific to Windows 7. To connect to a network using OS X Mac, see here. To connect using Linux, see here.*

Click on the Network icon on the windows menu bar. You should see a new connection that contains the name 'BeeWi'. If you can't see it, try refreshing the connection list.

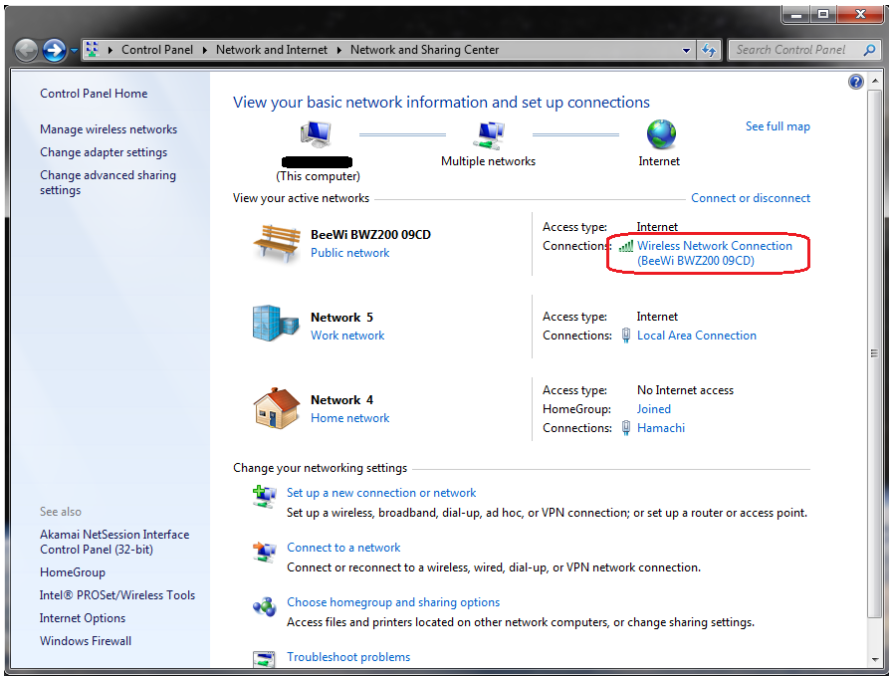


Once you have the connection available, try connecting to it. If it all goes to plan, you should now be linked with the Buggy.

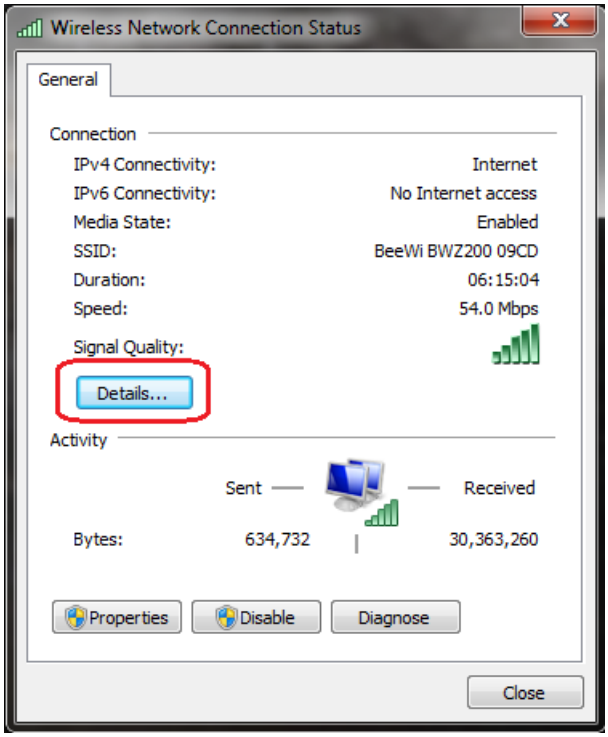
Getting the IP Gateway

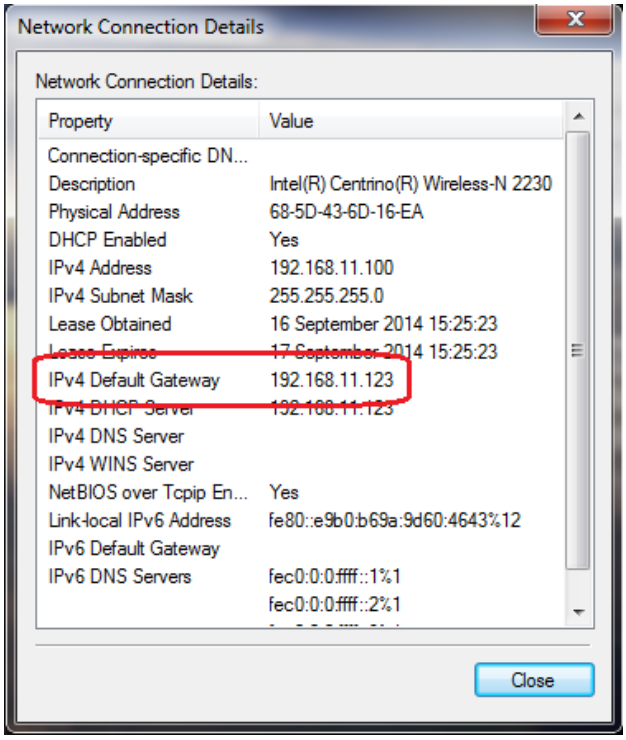
Once connected, you'll need to find out the Buggy's IP Gateway – this is used to control the Buggy and collect the video feed.

Click on 'Open Network and Sharing Center' within the Network List (*left*) and then click on the Buggy's connection.



Next, click on ‘Details’ to find out information about the connection. You should see the gateway address next to ‘IPv4 Default Gateway’.



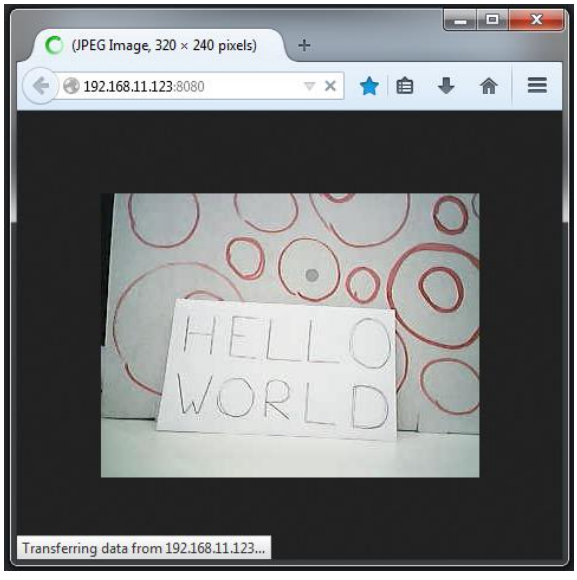


Video Feed via a Web Browser

One way to test that the connection is working is by loading up a web browser *(NOTE: This works for Mozilla Firefox 30.0, but not for Google Chrome 37.0 or Internet Explorer 11.0.96)* and using the IP gateway you just found, along with port 8080 (this is the port that the Buggy send the video feed along).

Type the following into the address bar: http://<IPv4_Gateway>:8080/ (replacing the gateway with your own, e.g. http://192.168.11.123:8080/)

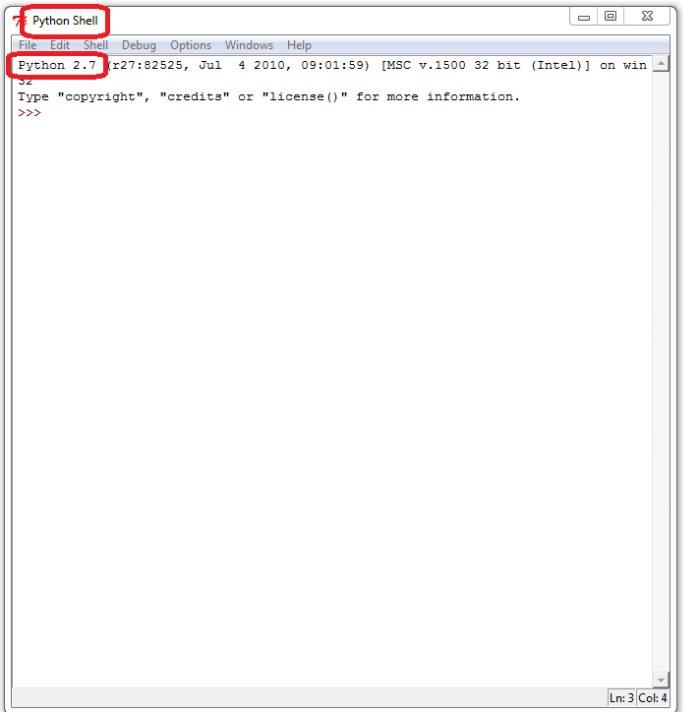
You should now see the Buggy’s camera within the web browser.

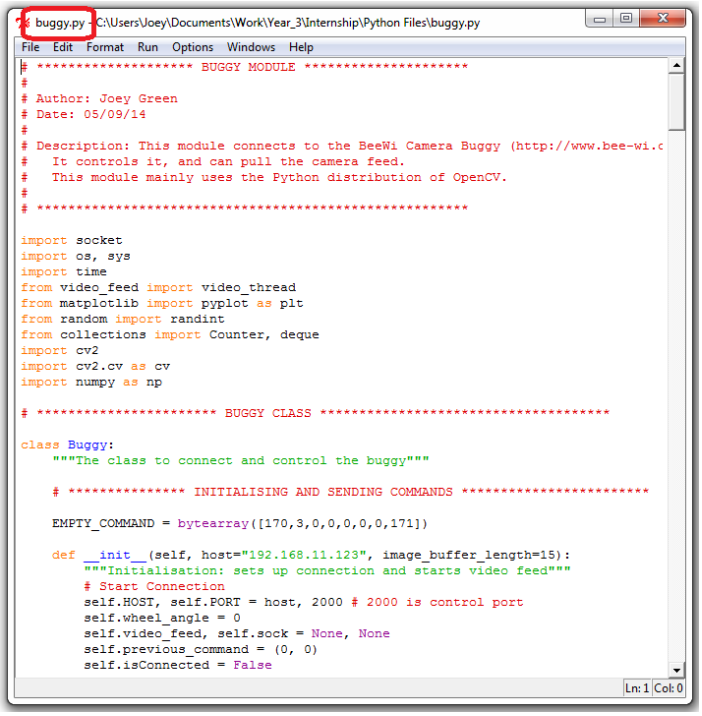


Controlling the Buggy using the SDK

Once the connection is all set up successfully and you have tested it by accessing the video feed via the web browser, you can then try running the *buggy.py* program. This program contains the core functionality for controlling the Buggy. Download this from the GitHub page, **as well as *video_thread.py*** – this contains the methods to handle the Buggy's camera.

To run this program, you will need to download and install the Python 2.7 IDLE[4] shell. Once loaded, you will see a window called 'Python Shell' as shown below. If you go to File > Open and locate the *buggy.py* you just downloaded, you should then see the program code in another window like below.





In order to connect to the Buggy, you'll need to use the IP Gateway you just found. Scroll to the bottom of *buggy.py* and find the line of code that reads 'buggy = Buggy()'.

Change this line to 'buggy = Buggy(host="<IPv4 Gateway>")'.

(e.g. buggy = Buggy(host="192.168.11.123")

If you click on the *buggy.py* program and go to Run > Run Module, you will run the buggy class, and it will instantiate an object of the Buggy class, helpfully called 'buggy'.

If you see a message that says 'could not establish connection', turn the Buggy off and on again, making sure it is fully charged, and reconnect to it via the previous steps.

If you see a message that says 'connection established', then you have successfully connected to the Buggy! Try it out by typing 'buggy.forward(1)' in the loaded shell. This tells the Buggy to move forward at the slowest speed for roughly a second. Make sure the Buggy has enough space to move!

[4] Python.org, (2014). Welcome to Python.org. [online] Available at: https://www.python.org/download [Accessed 17 Sep. 2014].

Writing your own Program

Additionally to the example programs (these are all open source, so you are able to play around with them), included is the *TEMPLATE.py* program. This is the starting point to writing your own program.

If you run the *TEMPLATE.py* without having edited it, it will basically use the Buggy as webcam – it takes in images through the camera and displays them in a small window (as DISPLAY is set to True by default).

If you find the comment that reads 'Do something with it', this is where you should start writing your program. It is within a 'while True' loop, so it will execute every iteration.

For example, underneath the comment write 'buggy.creep()'. This will move the Buggy forward a little bit each time it loops.

Have a look at the other programs available, and see what they do. A lot of them take the image from the Buggy, look for something within the image using OpenCV, and act accordingly.

If you're after a much simpler program, for instance you want to make the Buggy follow a certain path, just take out the 'while True' loop and put the series of commands consecutively within the program.

Python SDK

This project was intended to be used in outreach[5] events, to introduce children to programming. Python is a widely used, high-level programming language, which is very easy to learn. Python also offers an implementation of OpenCV[6], an optimised computer vision package. This is why Python was chosen to write the SDK.

The SDK includes the *Buggy* class, and the *video_thread* thread. On top of this, various Computer Vision algorithms have been implemented, showing to what extent the SDK can be used.

The *Buggy* class

When initialised, the Buggy can take two parameters – the IP of the Buggy (defaults to "192.168.11.123") and the image buffer length (defaults to 15). This is how many images to remember at once, and is used by the isStuck() method (*see below*).

The class sets up connection to the Buggy, returning the corresponding errors if it was unable to connect. The video feed thread is also set up, but it does not start saving images until the user explicitly calls the streamFeed() function.

Below are some functions that will be commonly used by programs importing this class.

forward(speed) and backward(speed)

These methods control the throttle of the Buggy. Each method takes values between 1 and 100 (if outside these ranges, they are capped respectively), and the corresponding command is then sent to the buggy.

turnLeft(degrees) and turnRight(degrees)

Methods to turn the Buggy's wheels left and right. As with forward and backward, these functions take values 1 – 100, and anything outside this range is capped. The *degrees* parameter is slightly misleading – it does not refer to exact degrees, just the value sent to the Buggy.

stop()

Stops the Buggy, equivalent to invoking forward(0).

straightenWheels()

Straightens the wheels, equivalent of invoking turnLeft(0) or turnRight(0).

creep()

A method comprised of moving forward, waiting, and stopping, very quickly. Allows the Buggy to move forward in a controlled manner.

backAway()

Same as creep(), but backwards.

[5] Wikipedia, (2014). Outreach. [online] Available at: https://en.wikipedia.org/wiki/Outreach [Accessed 10 Sep. 2014].
[6] Opencv.org, (2014). OpenCV | OpenCV. [online] Available at: http://opencv.org/ [Accessed 10 Sep. 2014].

randomisedRecovery(max_rotation = 100, max_speed = 30)

A method to randomly reverse the Buggy. It does this by randomly assigning a rotation between 1 and *max_rotation*, a speed between 1 and *max_speed*, and a direction (left or right). The Buggy then reversed in this speed and direction.

startVideoThread(), closeVideoThread(), streamFeed(), stopFeed(), displayFeed(), hideFeed()

Methods that communicate with the *video_thread* Thread (*see below*)

getCurrentImage(display = False, append = True)

Pulls the most recent image from the Buggy. If *display* is True, the image will be displayed in a window. If *append* is True, the image will be added to the image buffer (to be used in the *isStuck()* method).

addToImageBuffer(img)

If *img* is not None, adds it to the image buffer.

clearBuffer(), getAllImageBuffer(), getPreviousImage(), getImageFromBuffer(index)

Self-explanatory methods to get and set the image buffer.

isStuck(primary_t = 0.9, mean_t = 0.9, std_t = 0.01, d = 5)

The method to check whether the Buggy is stuck. It does this by first template matching the most recent two images, and if they are match less than *primary_t* then the method returns False (not stuck). If there is a match of more than *primary_t*, the function takes the image buffer and template matches the current image to all of the images in the buffer, and if the mean match is larger than *mean_t* AND the standard deviation is less than *std_t*, then the Buggy is deemed to be stuck and returns True. Else, the method returns False.

exitBuggy()

The method to cleanly close all sockets and shut down the video thread.

The *video_thread* Thread

This runnable class runs alongside the Buggy class and handles the camera feed. When initialised, the thread sets up conditions ready to stream the Buggy images. When running, if the camera feed is set to stream (via *streamFeed()*), the images are pulled via TCP and stored in a temporary variable, and when the tail of the image is found, it is then converted into a .jpeg image and stored in an image buffer.

streamFeed(display = False)

Connects to the Buggy through port 8080 and collects the camera feed.

stopFeed()

Stops the stream and closes the socket to the Buggy.

displayFeed()

Displays the video feed in a small window.

hideFeed()

Hides the windows displaying the video feed.

clearBuffer()

Clears the image buffer.

clearBuffer(), getAllImageBuffer(), getPreviousImage(), getImageFromBuffer(index)

Self-explanatory methods to get and set the image buffer.

Modules using the SDK

Currently, the following modules have been implemented:

pathFollower.py

This module allows the Buggy to follow a path of signs on the floor, in the form of directional arrows within a red sign. It can also stop when it sees a sign with a cross within it. These images are user defined variables – they all must have the same image size to work efficiently.

To do this, the Buggy first takes in its surroundings using its front camera. It can also warp the perspective of the camera using a user defined Homography matrix, allowing the Buggy to have a bird's eye perspective of its vision. It then takes the Chroma Red channel and searches for a circle, using the Hough Circle Transform[7].

If a circle is found, the content of the sign is cropped into a new image. This is then template matched with the images of the cross and the arrows. If an arrow is found, the Buggy looks at it for a set number of times, then takes the best direction and continues in that direction. The following parameters are:

DETECT_FLOOR – a Boolean that determines whether the Buggy will check signs on the floor, by using a Homography matrix to warp the camera perspective

DETECT_WALL – A Boolean that determines whether the Buggy will check signs on the wall.

MAX_RECOGNISING_COUNT – how many times the Buggy will observe a directional arrow before taking the average direction

MAX_TURN_STEPS – how many times the Buggy will continue to turn after following a directional arrow

PATH_CROSS – the file location to the image of the cross (the image to stop the Buggy moving)

NUMBER_OF_ARROWS – the number of arrows to use in the template matching. The minimum is three, being left, right and forwards.

PATH_ARROW_FOLDER – the path to the directory containing the arrows, labelled from 00.png (fully left) ascending

PATH_IMAGE_POINTS and ***PATH_OBJECT_POINTS*** – file locations to the image points and object points, to create the Homography matrix (only used if *DETECT_FLOOR* is True)

CHROMA_RED_THRESHOLD – the threshold used in the Binary Threshing on the Buggy's view, to find the red circle in the image

DISPLAY - the Boolean to display the Buggy's various images

[7] Docs.opencv.org, (2014). Hough Circle Transform — OpenCV 2.4.9.0 documentation. [online] Available at: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html [Accessed 16 Sep. 2014].

imageFollower.py

This module allows the Buggy to follow and back away from supplied images within red circles. It does this by finding circles within the Chroma Red channel of the Buggy's vision, and template matching the contents of any red circles found. If a match is higher than the threshold, the x position of the matched sign is taken and the Buggy drives towards or away from it (depending on the sign found). The following parameters are:

SIGN_FORWARD and ***SIGN_BACKWARD*** – the file location of the signs that tell the buggy to move forwards or backwards.

BINARY_THRESHOLD – the threshold to be used in the binary thresholding

MATCH_THRESHOLD – the threshold to be used when template matching the signs

DISPLAY – the Boolean to display the Buggy's various images

pictureTaker.py

A small module to save images taken from the Buggy's camera. All saved images will have unique names. The parameters are:

DISPLAY – the Boolean to display the Buggy's images

IMAGES_TO_SAVE – the number of images to save from the Buggy's camera

SAVE_LOCATION – the path to the directory where the images will be saved

WAIT_INTERVALS – the amount of time to wait between taking each picture

randomSearch.py

A module that implements Random Search. The Buggy randomly searches for a supplied image and stops when it finds it. It does this by continuing forward until it either sees the goal image (at which point the program terminates) or is stuck. If the Buggy is stuck, is uses the *randomisedRecovery()* method to reverse randomly and choose a new direction to search. The parameters are as follows:

DISPLAY – the Boolean to display the Buggy's images

PATH_GOAL_IMAGE – the file location of the goal image

GOAL_THRESHOLD – the threshold used in matching the goal image with the Buggy's current camera view

Appendix

To build an SDK to control the Buggy, the application was simulated on a computer and methods were built in Python to recreate the same control functionality, connecting to the Buggy via TCP.

An In Depth look at the Buggy

The application sends 8 byte commands to the Buggy to control its movement. The first two bytes are header values, the second byte controls the speed and direction (forward and backward), and the third byte controls the angle of the wheels rotation (left and right). The following three bytes are redundant, and the final byte is the tail.

Suppose the following command was sent to the Buggy (in Hexadecimal).

aa 03 7d 7f 00 00 00 ab

The *aa 03* would be the header, then *7d* would tell the Buggy to go forwards at full throttle, and the *7f* would rotate the Buggy's wheels fully to the left. The following empty bytes do not affect the Buggy at all, and the final *ab* is the tail of the 8 byte command. The header and the tail never change.

This data was found by intercepting the packets sent to the Buggy, and comparing the data sent when various commands were given.

Throttle

The Buggy's throttle byte takes values from *00* to *ff*. Any value within the range *10 – 7d* will tell the Buggy to go forwards, *10* being the slowest and *7d* being the fastest. Any value within the range *90 – ff* will tell the Buggy to reverse, *90* being the slowest reverse and *7d* being the fastest. The value of *00* will stop the Buggy moving.

The forward() and backward() methods use values in the range of 0 – 100 for simplicity, and this is converted to a value within the available range. i.e. 0 – 100 forward corresponds to *10 – 74*, and 0 – 100 backward corresponds to *90 – f4*.

Turning

The Buggy's rotating byte takes values from *00* to *ff*. Any value within the range *04 – 7f* will rotate the wheels left, *04* only turning slightly and *7f* rotating fully left. Any value within the range *84 – ff* will rotate the wheels right, *84* only turning slightly and *ff* rotating fully right. The value of *00* will straighten the Buggy's wheels.

Video Feed

In order to control the Buggy, you must connect to its IP Address (e.g. 192.168.11.123) and send the data over port 2000. Port scanning the Buggy revealed three different ports to be open: 2000 (the control port), 8080 (usually an alternate HTTP port), and 23 (a Telnet port). The Telnet is password protected, and no common user-password combinations work.

When a user connects to port 8080, the Buggy immediately returns the video stream, in the form of TCP segments. These can be stored in a buffer, and when a whole image has arrived it can be regexed and stored as a .jpeg image.

Limitations

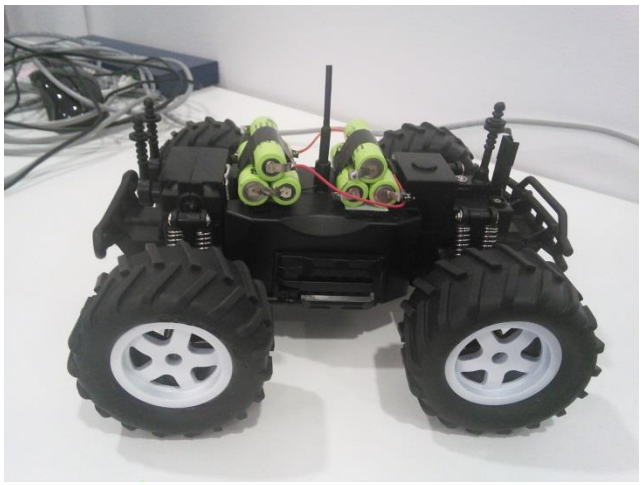
The Buggy does not have a built in scheduling mechanism, so if two commands are sent to it within a short space of time (less than 0.5 seconds apart) it will freeze for a few seconds, before resuming from the most recent command sent.

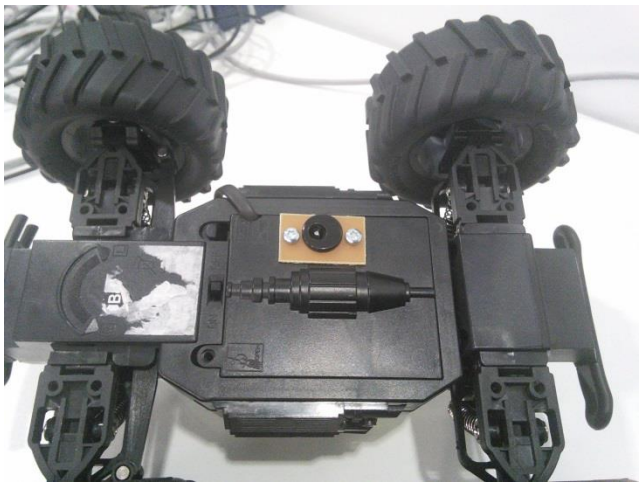
The Buggy was also built for speed. This means that although its top speed is exceptionally high, getting the Buggy to move slowly is very challenging. The *creep()* method has been implemented in order to get the Buggy to move as slow as possible, using a mixture of moving, waiting and stopping mechanisms.

The camera supports a resolution of 320x240 – this quality is fairly poor when using the Homography matrix to get the Birds Eye Perspective of the camera view.

Modifications

When the SDK was under development, the Buggy running out of charge became quite an issue. To counter this, additional battery packs were added, and a charging port was fitted on the underside of the Buggy to allow for 'on the fly charging' so testing could be done while the Buggy was still charging.





As mentioned before, there are three unused command bytes. These could potentially be used to control an external peripheral – for example, one could add LED lights on top of the Buggy to show a various state the buggy is in.

The camera has a resolution 320x240, but this could potentially be upgraded to a 640x480 if one wanted to do more advanced computer vision tasks that rely heavily on the quality of the images.

About the Developer

Joey Green is a Computer Science student at Durham University.

This project was part of a Summer Placement with Durham University.

Contact him at joseph.green@durham.ac.uk .