A model Driverless car using computer vision

12RobinC

Fulford School  [Company address]

Contents

[Analysis 2](#_Toc5980687)

[The Problem 2](#_Toc5980688)

[How can it be solved using computational methods? 2](#_Toc5980689)

[Methods and explanations: 2](#_Toc5980690)

[Problem recognition 2](#_Toc5980691)

[Decomposition 2](#_Toc5980692)

[Divide and conquer 3](#_Toc5980693)

[Abstraction 3](#_Toc5980694)

[Stakeholders 3](#_Toc5980695)

[Research of Existing solutions 6](#_Toc5980696)

[Essential features 7](#_Toc5980697)

[Limitations 8](#_Toc5980698)

[Hardware and Software requirements 8](#_Toc5980699)

[Stakeholder requirements 8](#_Toc5980700)

[Success criteria 10](#_Toc5980701)

[Design 11](#_Toc5980702)

[Decomposition 11](#_Toc5980703)

[Structure 12](#_Toc5980704)

[Algorithms 13](#_Toc5980705)

[Usability 14](#_Toc5980706)

[Variables, Data structures and classes 16](#_Toc5980707)

[Validation 17](#_Toc5980708)

[Developing 18](#_Toc5980709)

[Iterative development 18](#_Toc5980710)

[Prototypes 18](#_Toc5980711)

[Prototype 1 18](#_Toc5980712)

[Prototype 2 23](#_Toc5980713)

[Prototype 3 29](#_Toc5980714)

[Prototype 4 31](#_Toc5980715)

[Addressing the failed tests 34](#_Toc5980716)

[Prototype 5 35](#_Toc5980717)

[Addressing the failed tests 37](#_Toc5980718)

[Prototype 6 39](#_Toc5980719)

[Evaluation 40](#_Toc5980720)

[Post Development testing 40](#_Toc5980721)

[Success criteria 43](#_Toc5980722)

[Usability Features 45](#_Toc5980723)

[Future Development 48](#_Toc5980724)

[Limitations 48](#_Toc5980725)

[Bibliography 48](#_Toc5980726)

# Analysis

## The Problem

Computing is a fast growing industry and it is important that children get involved in computing now so that in the future they have the opportunity to work in this innovative field. The problem is that for young people with no experience programming or any higher level computing education, coding and computing in general can seem overwhelming and confusing and it is difficult to see the immediate application and positive feedback from coding. The idea is to produce an interactive product that can encourage school children to get involved in STEM subjects.

My project is to produce a physical model that shows off the applications of computer vision and programming to a younger audience to get people interested in computing. I will make a small model car, driven by a computer using computer vision to identify objects and follow them. This requires a camera, a car, a computer vision library, and code to control the car. I will use iterative development to create the main code. I will focus on a procedure, complete a prototype, and move forward, procedure by procedure.

## How can it be solved using computational methods?

The problem can be solved using computational methods because. The solution will be software that connects to both a car and camera to detect objects and control navigation. These processes need a computer to work.

### Methods and explanations:

#### Problem recognition

The problem I am trying to solve is how to reliably identify and track an object and have that method interface with a model car that eventually uses algorithms based on the tracking to navigate towards a decided item. To control the car in software and then have algorithms that depend on the location of the tracking object.

#### Decomposition

The problem I have identified can be broken down into more manageable chunks, which should combine into the finished solution.

* Connecting a camera to stream video to the program
* Add a mask that isolates the desired object
* Using a computer vision library to track the object’s movement
* Using the tracking info to move the car

#### Divide and conquer

The chunks are split further into individually manageable tasks. Doing each part separately producing a series of separate prototypes that each meet a specific goal.

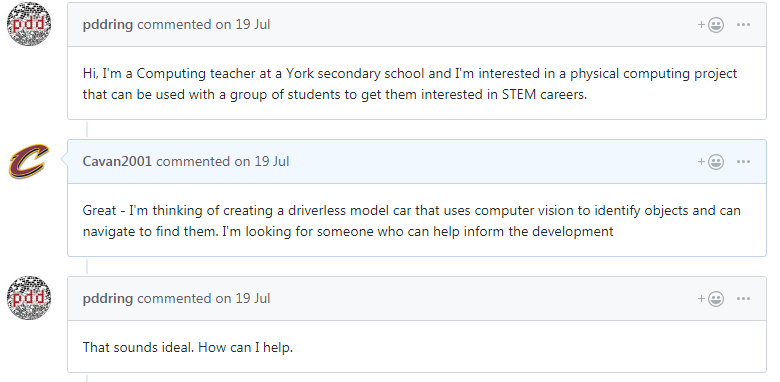
#### Abstraction

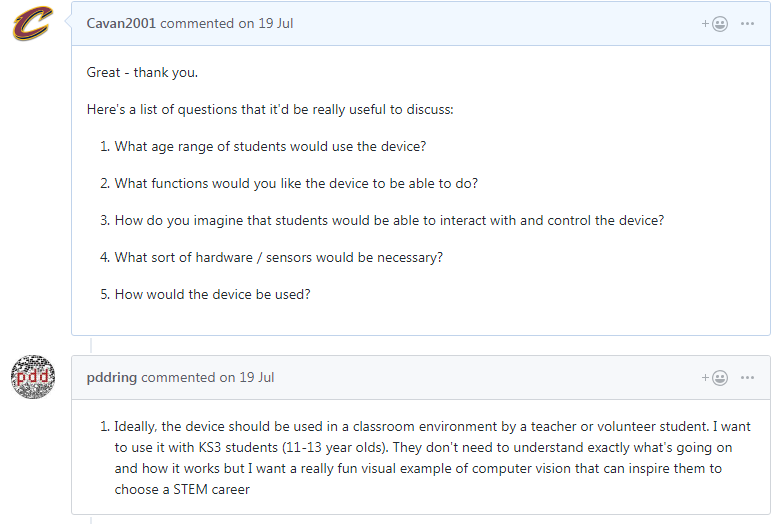
The car does not need to constantly make corrections or know about every single coordinate of the object. The car should only care about general segments of the image and have an algorithm for each section; only the centre point of the object should be tracked. When isolating the image only the outer boundaries of the object should matter, anything inside is useless. The car also will not actually calculate the distance between the object and the car; this will be abstracted by knowing the size of the object and the position of the objects centre point as the car gets closer.

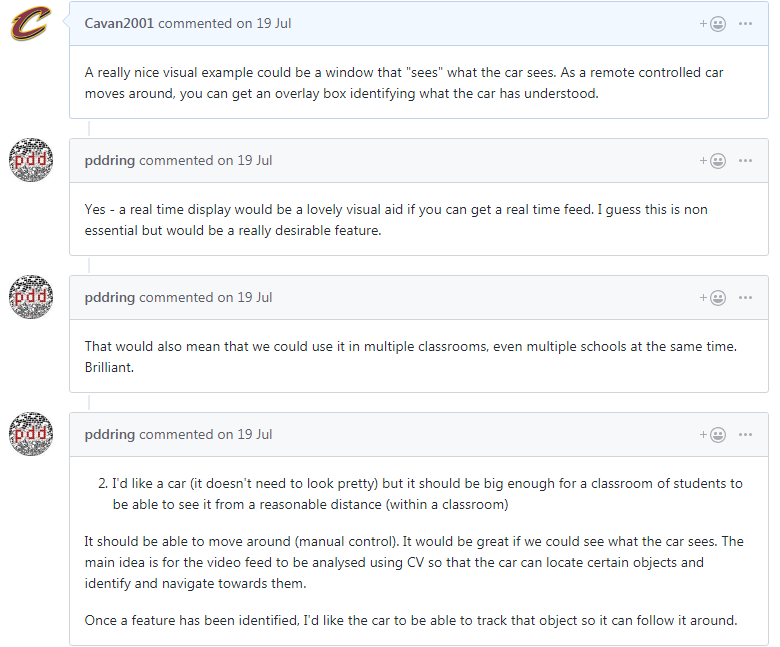
## Stakeholders

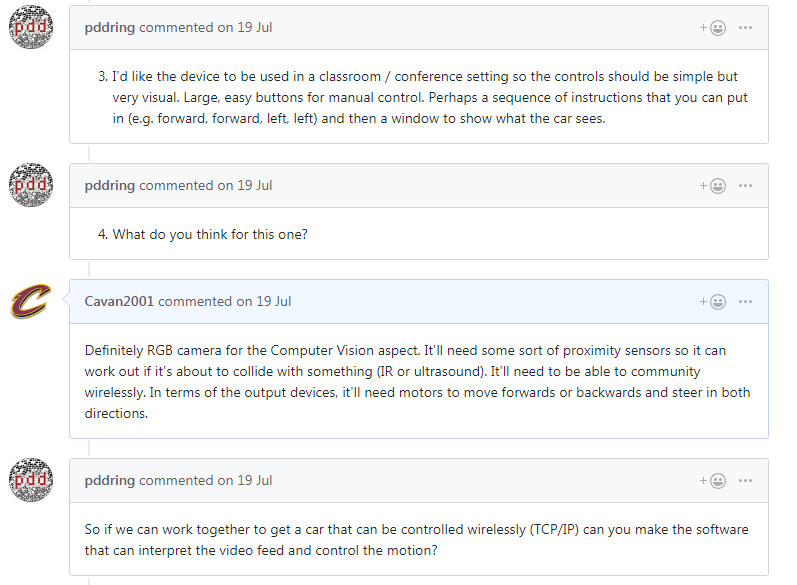
A possible use case for my product could be a STEM teacher. Demonstrating how computing, maths and computer vision algorithms combine to create an interesting and engaging product. People interested in learning how to implement computer vision in their projects may also be a possible stakeholder, this product could be a window into how to implement computer vision. RC enthusiasts may enjoy this product as the car will be a modified RC car.

**A conversation with a stakeholder Mr P. Dring**









**What I learned**

I learned that the use case of this product would be a presentation and that the product needs to be interactive for a group of students, I need to have a visualisation of the computer vision seeing what the car sees. It also means I should keep the UI simple and clear, a class should be able to read and understand the window when looking form a short distance. It needs to be quick and easy to connect for a presentation. The children should also be able to control the car manually. I need to have instructions for the demonstration to aid clarity and usability.

**An interview with student Harry Brown**

Me: “What things would you want from an interactive presentation using a driverless model car?”

Harry: “To be able to have the students control it ourselves.”

Me: “Manually you mean?”

Harry: “Yes like the arrow keys or something”

Me: “And what would you like from the computer vision aspect of the program”

Harry: “I want to be able to pick the things I drive to. And be able to pick them easily so stuff should be clearly labelled or instructed”

Me: “In terms of visual representations of computer vision in the window what would you like?”

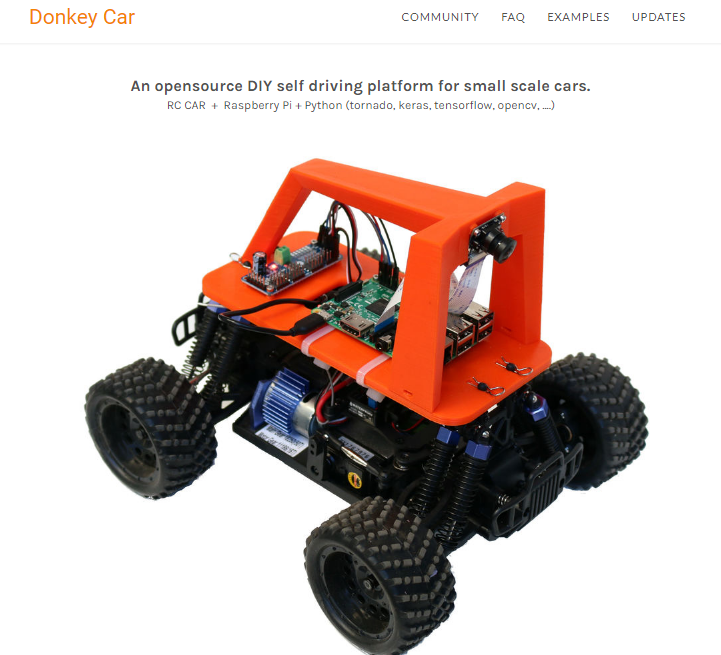
Harry: “it would be cool If I could see the camera feed and then like a box or circle around the object”

**What I learned**

The students should be able to control and use the program themselves, using the keyboard. They want to be able to pick the things that the program tracks and drives to. The window should show the video feed with a box over the object showing what the program is tracking.

## Research of Existing solutions

**Donkey Car**

An example of an existing solution to my problem is the donkey car. It uses a raspberry pi, with a raspberry pi camera, and OpenCV to detect the lines of a track and drives the car around the track. This method tracks the edges of a line on the tack and stays in the centre of the line. This project is primarily about following a defined track and tracking a white line against a well-contrasted black floor; the donkey car is open source, has its own library and offers a kit to construct it. This makes it easily accessible and appealing however, it does not quite solve my problem; I need to be able to track and identify a specific object. Their system is also modular; the kit can be fit to “any RC car” meaning it has wider accessibility. It also has a smartphone app, which is a great idea just not one I can include on top of my other features. I also researched OpenCV and found many tutorials and support, this will make my problem easier to solve.

**Key points**

High contrast between tracking object and background, this means I ideally should track brightly coloured objects that contrast the colours in the room.

Easy to use, intuitive UI. The end user should have no problem understanding the program and how to operate the car.

**Antonin RAFFIN’s “autonomous racing robot with an Arduino, a raspberry pi and a pi camera”**



Another car that uses the pi camera, this time with an Arduino. In addition, codes in c++. This project is more similar to my solution however as it is done by a single person, this allows me to see how the general decomposition and development works. Again, however, this project tracks a line and follows a

Specific path. They use colour to detect the line, which I think is a simple and efficient solution to the masking and identification problem. His solution however uses machine learning, which I think is too ambitious and not needed for our brief.

**Key points**

The method of contour detection is the way I plan to track the desired object.

Other than that, this solution is not particularly applicable because this solution uses machine learning.

**The library**

In terms of the computer vision library I will be using. I am looking for few things; It needs to be in a language that I somewhat understand and be able to interact with the other code that I write to control the car.

The most widely used computer vision library is OpenCV, and the latest iteration of the library is OpenCV 4.0, however it uses python primarily so I want a shell for OpenCV for either vb.net or C# these are the languages I am most comfortable with.

After some research, I found that the primary C# shell for OpenCV is EMGU CV. It has all of the functionality of OpenCV in a language that I understand.

## Essential features

**Program idea after research and considering stakeholder input:**

My stakeholder has identified man essential features of the project:

-The project must be interactive as it is designed to be an educational tool.

-We need a window that shows what the camera on the car is seeing, and the objects that it is detecting. It was described as non-essential but desirable I think it is very doable.

-It needs to also be able to be controlled manually but also autonomously using object detection and algorithms.

- It should have a follow function. It needs to be easy to use and see. The camera should be an bgr camera for object detection but additional sensors such as ultrasound sensors may be needed for proximity.

- It needs to operate wirelessly.

Therefore, the main window will contain a preview of what the camera sees, a easy to use method of selecting the item you wish to track, simple buttons that switch between manual and computer control, and buttons that allow you to connect to the car. The UI should be easy to use and the buttons should not be confusing or get in the way.

## Limitations

The extent of the computer vision and object detection will be limited. Cannot generate libraries based on specific objects using harrcascades, as this would take more time than I have to complete this project, so the computer vision will be based on colour, because of this it will mean in variable light conditions the reliability of the detection will decrease as colour values will change.

Another limitation is the range of control I have with the car, depending on the car I use the angle of turn or rate of speed may not be accessible and therefore the amount of control the car will have is limited.

The range of the microbit and camera radio frequencies mean that the car can only be operated within the room the computer is in.

I cannot fit multiple cameras on the car, so if the object falls outside of the cameras field of vision then the object will be lost.

## Hardware and Software requirements

**Software:**

-**EMGUCV** an open source, computer vision library with thousands of objects stored. This is a good option because it is free, well tested, and unlikely to have bugs and there are many resources online for help and inspiration.

- **C# interpreter**, I will be coding in c# because it is a language I am comfortable with, is efficient, and high-level.

-**either windows, Linux or mac** operating systems supported by emgu.cv

**Hardware**:

**Two microbit controllers -** will be used to allow the car to communicate with a nearby computer. This is a viable option because they are available to me at school and relatively inexpensive.

**A hobby grade FPV camera -** to attach to the car, this will be the camera used by my program for object detection,

**A software controllable RC car –** needs to interact with the computer and my tracking program.

## Stakeholder requirements

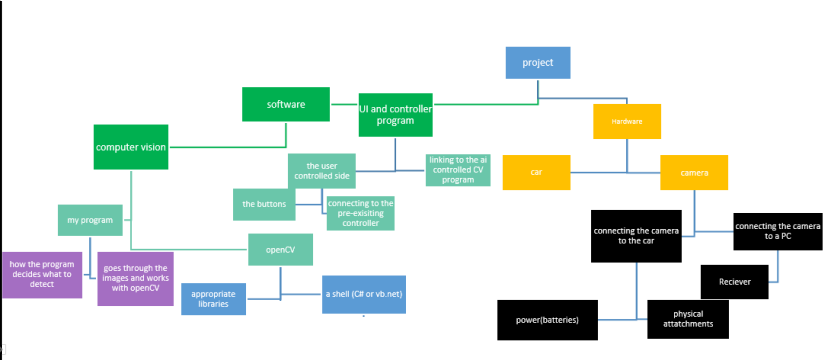
|  |  |
| --- | --- |
| Requirement | Justification |
| Intuitively designed UI | Kids need to be able to control the program with little to no knowledge of how it works |
| Clearly demonstrated video feed | Showing what the car is seeing and currently tracking will help students visualize the computing going on. |
| Manual control | The ability to stop the computer from controlling the car and revert to manual control. |
| Clear way of selecting a tracking colour | The process of isolating the colour you want track should be as fast and as easy as possible. |
| Algorithms that navigated to the desired object | The car should be able to navigate to a desired object and indicate to the user that the navigation is complete. |
| Follow function | A function that continues the navigation method toward a moving object and continues moving until instructed otherwise. |
| A wireless FPV camera | Needs to be wireless to be fit onto the car and needs to be high enough resolution and frame rate to be adequate for the tracking algorithm. |
| EMGUCV | The computer vision library that will deal with the object detection and tracking |
| Car control program | A simple program that allows you to control the car using a nearby computer. |
| Instructions page | A window introducing the user to the program and instructions on how to connect the camera and the car. And a control scheme for the manual controls |

## Success criteria

|  |  |  |
| --- | --- | --- |
| Criteria | Explanation of the criteria | How I will meet the criteria |
| 1. Manually controllable “↓” “↑” “←” and “→” | School students need to be able to easily control the car to do a first pass of the room. It will keep them involved as well. | A basic manual car control set up will be done before any of the computer vision aspect to ensure that this criteria is met |
| 1. Large and clear car model | The car needs to be easy view in a classroom setting, the car cannot be minuscule. A whole group of students need to be able to see the car and understand what it’s doing | When testing with stakeholders I will take a survey and ask if the car was large enough |
| 1. Fast and Responsive. | Should not have too long wait times, to set up and calibrate. If it takes too long students may become disinterested | I will use as many programming techniques as possible to make my code run efficiently, I will try and keep my camera connection low latency and I will try to have the car go as fast as it can while still being safe |
| 1. A live feed that allows you to “see what the computer sees” | Showing both the live camera feed and the tracking box for the object would be a good visualizer | Use picture boxes in windows forms to display the video feed and paint shapes onto of the image showing the current image being tracked. |
| 1. Intuitive and clear UI | Kids with little to no experience should be able to use the program without much prompting. | Use as little buttons and solutions as possible. A clearly labelled and instructional form (window). |
| 1. The selection of a tracking colour should be simple | Should not be clear how to go about selecting the colour of the desired object. Only changes should be fine-tuning. | I will do this by creating an array of colours and their HSV values. When a user selects the colour, the mask should change to accommodate these values. Fine-tuning will be needed to adjust for lighting and other variables. |
| 1. Simple button that initiates the navigation | One button should be used to initiate the navigation to an object to keep the design clear and intuitive | Have the algorithm to navigate be split up into other more manageable algorithms that are activated by one button |
| 1. Instruction window | A window that tells the user how to connect to the car and how to manually drive it. | Have a separate windows form that contains the control scheme and instructions open before the main program, then when the user clicks a button the instruction window closes and the main form opens |
| 1. The ability to interrupt the navigation | Should the navigation go wrong or if anything becomes damaged, the user needs to be able to interrupt and stop the navigation without closing the program | The manual control button should override the navigation function, once the user decides to continue navigation; they van again select navigation and proceed. |
| 1. Function that follows a moving object | The option to start tracking and following a moving object, is more interactive for students | Continuation of the navigation function that continues until interrupted. |

# Design

## Decomposition



I have decomposed the problem into smaller more manageable parts that can be tackled using iterative development. Each box represents a problem to be solved, or a feature of the project that requires research. The implementation can be split up again into manageable goals that will combine to create a complete solution:

## Structure

**Camera connection**

I have an fpv camera, which works by sending out radio signals on the clearest frequency it can find. In addition, a receiver, which plugs into the pc and scans frequency until it finds the frequency being broadcast. I will use the computer vision library EMGU to stream the video onto the form. I will grab the current frame of the capture and add to a picturebox. This needs to be done several times per second. In addition, should be regulated to the fame rate of the camera to reduce frame loss. I need the highest frame rate possible, because it takes time for the car to process and react to the information given. In addition, the image processing will probably reduce the number of frames per second itself.

**Car connection**

The car I am using has a microbit in it. The microbit can be used to control the car. I need to send signals to the microbit from the PC. The microbit can send and receive radio signals at 2.4 GHz. it is easier to have one microbit connect with another than to try and connect the computer directly to it. You can send a list of simple commands that control the car e.g. speed, direction, lights On/Off.

The code that sends the radio signals and initiates the connection is already written, I will need to connect that to my project and create a new object of that class, then write code that controls the car.

**Car Movement**

My solution to the “locate and navigate to an object” problem is as follows:

1. OpenCV works by drawing a rectangle around an object once detected.
2. So the car will position itself so that the rectangle is in the centre of its vision
3. Once centred the car should accelerate, continuing to keep the object in the centre of its vision. (making corrections)

The idea is to split the image into a 3x3 grid and based on the current segment of the grid the object is in a different car control algorithm runs. Therefore, the intention is to keep the object in the centre segment and to accelerate to toward the object.

The follow function would be a continuation of the navigation function, so that it does not stop navigating once it gets close to the object. If the object moves away from the car, it will continue to drive towards it.

## Algorithms

**Object detection algorithm**

Object\_detected = False

Findcontours(image)

FOR EACH contour IN image

IF contour.size > 5 THEN

Contour. Draw.bounding\_rectangle(object.X, object.Y)

Object\_detected = TRUE

NEXT

**Car Turning Algorithm**

**Computer:**

IF obect.centre = centre\_segment THEN

Accelerate ()

Else IF object.centre = left\_ sement THEN

TurnRight ()

ELSE IF object.centre = Right\_segment THEN

TurnLeft ()

END IF

**Manual:**

IF Key\_Down.Up THEN

Car.speed = 10

Car.direction(forwards)

END IF

IF Key\_Down.Down THEN

Car.speed = -10

Car.direction(forwards)

END IF

IF Key\_Down.Left THEN

Car.speed = 10

Car.direction(Left)

END IF

IF Key\_Down.Right THEN

Car.speed = 10

Car.direction(Right)

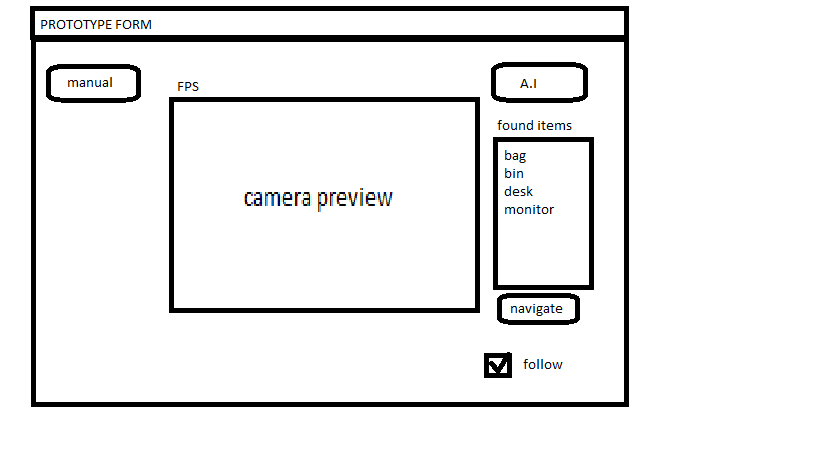
END IF

**Linking algorithms and structure**

The camera connection works with getting the preview in the form. Once the connection and preview are finished, I can implement the object detection. Using the video feed obtained in the first iteration, I can begin to isolate the colour values from the image and add sliders to pick colours. From this, I can use EMGU to track the contours, draw a rectangle onto the image, and display that on the preview. Using that I will develop the manual, car controls. Once manual control has been implemented I can modify those controls to work using the camera, for this I will need to have algorithms specific to s certain segment of the screen this is where I will use my car movement algorithms.

## Usability

My proposed screen designs, I want them to look fun and engaging for the students but also professional enough that the product is clear and a good advertisement for computer science. The buttons need to be large enough for the class to see, as does the preview, they need to be intuitive and interactive. The instructional screens text should be large and legible. All the processes of the form should be quick and easy to figure out for a student trying to use the program.

**Main form:**

Colour picker

Colour picker

The camera preview will show “what the camera sees” it will also show the current object being tracked with a green rectangle around the object.

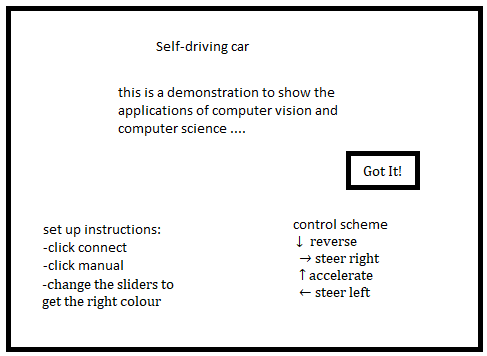
The “Navigate” button will switch the car controls to the computer and the user will no longer be able to drive the car, until the user presses the manual button, which allows the user to control the car with the arrow keys.

The colour picker allows the user to isolate the colour they wish to track before they begin controlling the car. In addition, the follow function means the car will continue to follow the object.

**Links to the success criteria**

There are few buttons on the window; the preview is large and clear. The form is simple and yet meets all the requirements of success criteria, the user is able to select the colour of the object. Can see what the camera sees, can both manually and allow the computer to control the car and select the follow function.

**Instructional form:**



The form itself only has a button and some text. The button closes the instructional form and opens the main form.

**Links to the success criteria**

This adds the instructional window criteria. It ensures the user knows how to use the program before they are able to access the main form. The simplicity of this form is to make sure the text can be as big and as clear as possible. This window can also act as the splash screen for the project, advertising and directly addressing the students that are viewing the demonstration.

## Variables, Data structures and classes

|  |  |
| --- | --- |
| Data structure | function |
| form **Preview** | Displays camera feed in the form and handles image passing and video capture |
| Form **instruction** | Has a welcome message along with a set of instructions |
| Class **Vision** | This class will handle the entirety of the object detection. |
| List **identified contours** | A list of all of the currently identified items in the room |
| Class **Controls** | This will contain all the communications with the car and the ability to control it |
| Function **A.I navigation** | The code that actually controls the car and moves it to the selected item. |
| **imgBGR** variable | A matrix that holds the original BGR image |
| **imgHSV** variable | A matrix that holds the converted hsv image |
| **imgGREY** variable | A matrix that holds the hsv image with the applied mask in black and white |

## Validation

The user won’t really input much in my solution, they will press buttons, use sliders and press the arrow keys, the buttons and sliders are robust as they are part of Visual studio, the only thing is if they are pressed at the wrong time, for instance id the user presses the navigate button when no object is selected the code may need an object to run the code and crash, therefore I will make sure that code only tries to run if the user has first selected an object.

It is possible that if the user tries to press the arrow keys before connecting to the car, code will ru that tries to control a car that isn’t there so may lead to a crash, I will make sure that this code only runs after the car has been connected to the program..

**Test data**

|  |  |  |
| --- | --- | --- |
| test | Expected outcome | Reason for test |
| Pressing the connect button | The camera preview starts and displays an FPS timer | To check if the camera connection and preview is functional |
| Using the arrow keys in manual mode | The arrow keys should control the car | To see if a connection between the two microbits has been established and whether the controls are functional |
| Moving the slider | Should change the colour values of the mask to isolate a colour | To test if the colour picker is functional |
| Try and connect when no camera is connected | Messagebox telling the user to connect the camera | To avoid crashes if the camera becomes disconnected or if someone tries to connect too early |
| Try to use A.I navigation while in manual control | Message telling you to enable A.I function | To avoid crashing |
| Place similar looking items in front of the car during navigation | Ignore the similar item | To improve the accuracy of the A.I |
| To click the navigate button | The car locks the user out of manual control and begins being controlled by algorithms alone | To test whether the user can still control the car during computer control and to see if the computer control functions properly. |
| To click navigate without any items selected | A message informing the user to select an item | To avoid crashes and test the selection algorithm |

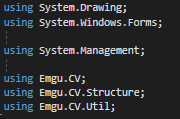
# Developing

## Iterative development

## Prototypes

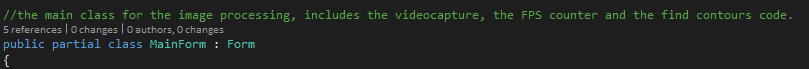
## Prototype 1

I need to create a new windows form in c# and add the relevant imports for this iteration

These are the imports I need for this prototype; the system drawing is to draw images to the picturebox from a matrix.

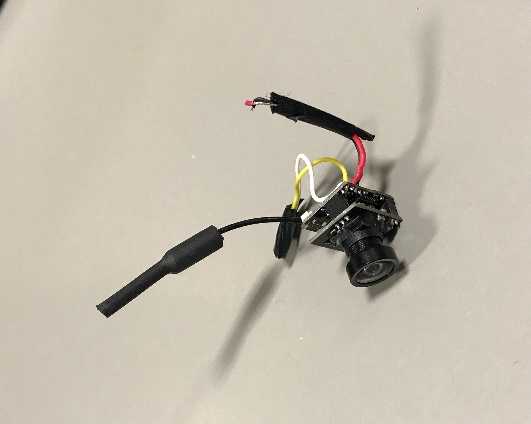
The windows forms is to control and interact with the UI

The EMGU imports are the computer vision library imports I will be using for the entirety of this project.



I then make the first class for the main form that will tackle the image preview, processing and detection. As there will only be two form’s in the project and this will be the main form I will name this “MainForm” and the future instructional form “instructions”.

I also then add a toolstrip to my UI, and in that toolstrip I add a play button that will connect the camera to the program and output a video stream.

This is the camera and receiver I am using:

The idea for the preview is to use emgu to stream video from the camera to the pircurebox in the form; I need to paint each frame into the box. When finished I can use this preview and the video stream to start adding image processing.

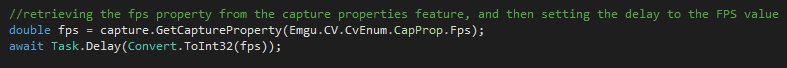
The camera outputs a radio frequency and the receiver scans through frequencies until it finds the camera. The version of OpenCV I’m using (EMGU) allows you to start a video capture using a camera connected to you PC.

Then, once the video capture has started, we need a matrix that can hold the current image of the video capture



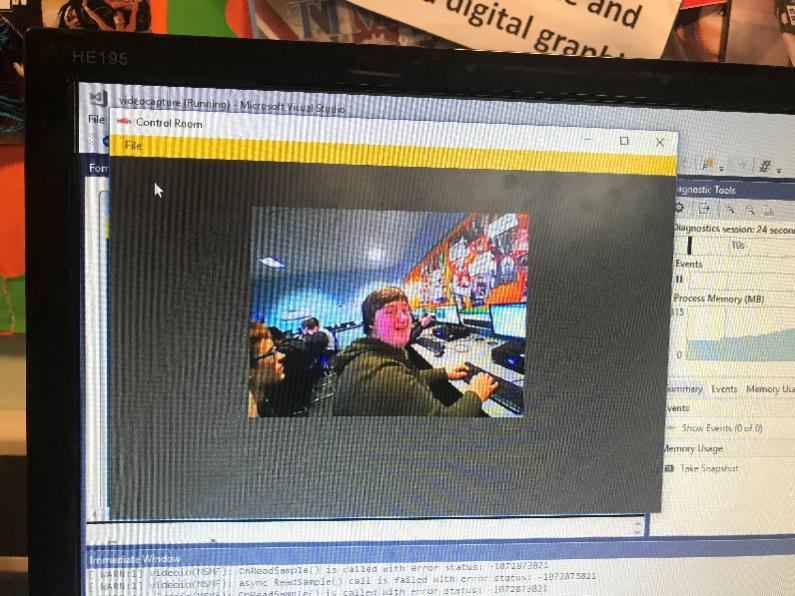
Next, you need to update your picture box to show the current frame.

I need to do this every time a new frame is passed to the “videocapture” to do this I need to find out the property of how many frames are passed to the capture per second. EMGU has a capture properties method that includes FPS. Once I obtain this value, I need to set the delay of the task that paints to the picture box to the time it takes to get a new frame:

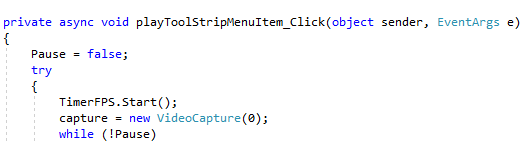


“convert.toInt32” is a function that converts the capture property FPS into a 32bit signed integer that can be used as a time delay for the asynchronous sub routine.

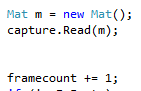
This results in an image that looks like this:



Next I wanted to add a Frames Per second (FPS) counter on my UI, this would allow me to monitor how regularly the frame was updating and how well it was processing the images. I can’t use the FPS capture property as this only gives the number of frames passed to the videocapture peer second, not how many frames pasted to the picturebox per second, this is important as the FPS will change depending on how much Image processing is going on. I did this by adding a timer that starts when the camera starts drawing to the picture box.



Integer “variable” increments every time a new frame is sent.



The timer ticks every second and so the number of frames counted resets every second so we output to a label that updates the FPS.



This is the extent of the first prototype, I am able to display the feed from the FPV camera to my program, and show the FPS.

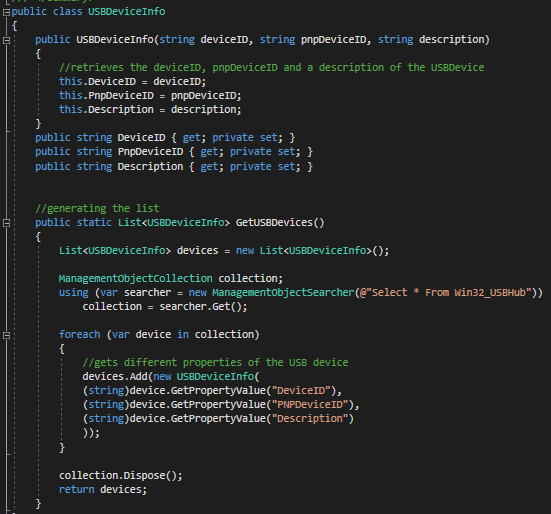
|  |  |  |
| --- | --- | --- |
| test | Expected outcome | Actual outcome |
| Attempt connection with no camera connected | Message box that tells the person to connect a camera | **Fail** No message the subroutine still runs but no image appears. |
| Unplugging the camera unexpectedly | The picturebox to stop updating and wait for the camera. | **Fail** Crash, runtime error |
| Connecting with the camera plugged in | Video stream in the preview box with the fps counter working properly | **Pass** |

**Addressing the failed tests**

I want to add validation to check whether the camera is connected because at the moment a message box does not display when the camera is not connected and the connection code still runs and tries to update the FPS counter.

I want to do a check to see if the camera is connected. To do this I need to know which devices are connected to the PC and if the camera is one of them. Therefore, I started to research how to enumerate the connected USB devices. Many of the resources I found were very convoluted. However, I came across a relatively simple and easy to understand method.

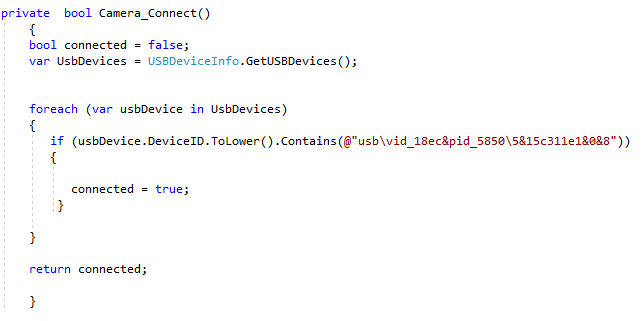
<https://stackoverflow.com/questions/3331043/get-list-of-connected-usb-devices>



This method works by creating a list of all of the devices connected using the “Win32\_USBHub” and for each item in the list of devices, we get a DeviceID, a PNPDeviceID and a Description of the device.

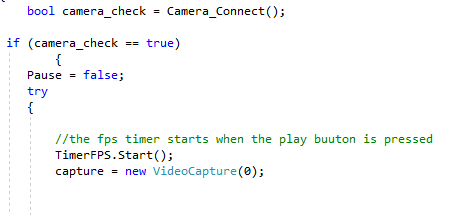
We can then use this class to grab the info on the USB devices information and check to see if the information matches that of the camera.

This lead me to create the function Camera\_Connect():



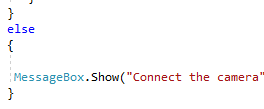
I make a variable USBDevices and have it equal to the GetUSBDevices() function. Then for every USBDevice connected to the system it checks to see if the DeviceID is the same as my camera’s specific deviceID. In addition, if the camera is connected and found. The Boolean “connected” is changed to true. Then we return the Boolean value.

This function is called when the play button is pressed:



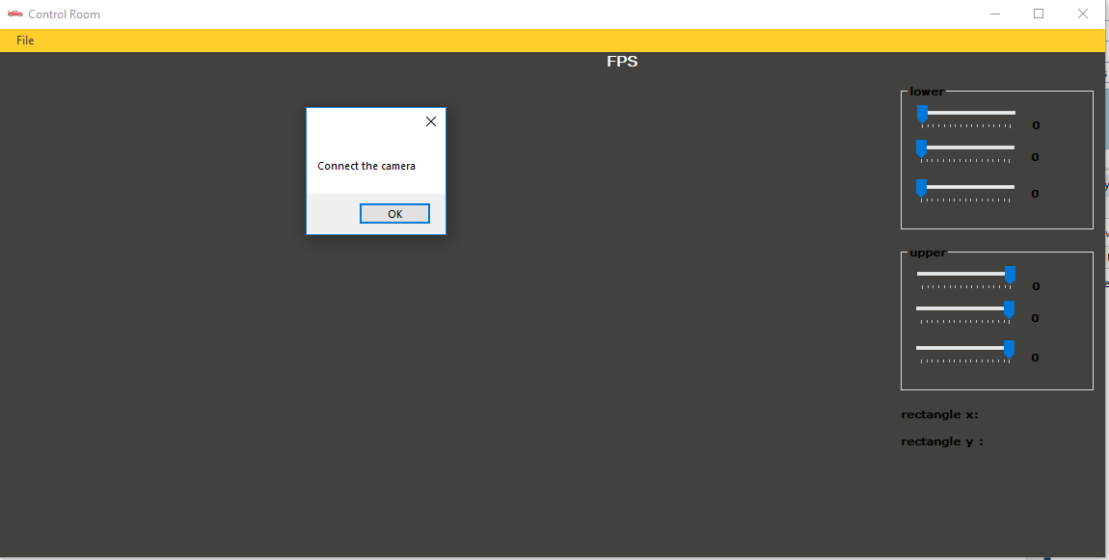
We run the check and if the camera is connected, the camera display code starts.

And if not:



We tell the user to connect the camera.

Now, when we try to press play with no camera connected we see this:



This also fixed the unexpected unplug issue. If the camera is disconnected, images are no longer pushed to the picturebox and there is no runtime error.

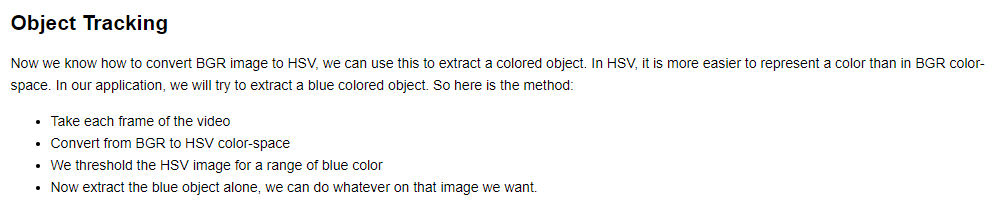
**Linking back to the success criteria**

Success criteria 4 and 5 are met in this prototype as we add the preview image that “sees what the camera sees” also add and FPS counter. The UI at this point is clear and easy to use. A single button in the file toolstrip. I uses a toolstrip as it enables you to hide certain buttons and prevent them from cluttering the form. The validation also means that there is no current way to crash the program, and any invalid inputs are caught and the user is notified.

## Prototype 2

Now that I have the camera outputting, a video feed. I want to isolate a certain colour in the image. For example a red book.

I need to create a mask on top of the image to allow only certain values in the right colour space to be recognised. A mask is a range of values in a certain colour space that are accepted, for this you need a minimum and maximum value, or upper and lower band of colour values that isolate the accepted colours to only the ones you wish to identify. To go about implementing this I researched the best colour space the image should be in. by default in emgu images are set to the bgr (blue green red) colour space. In my research, I found:



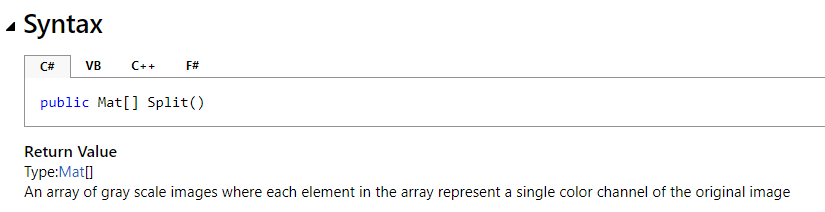
<https://docs.opencv.org/3.4.2/df/d9d/tutorial_py_colorspaces.html>

Therefore, I will convert the image into the HSV colour space. Give the image a “threshold” for the colour we want to isolate and somehow track that.

I looked up how to do that in C#. Turns out you can explicitly declare the image in the HSV colour space. Like this:



After we have done this, we want to be able to control each of the three channels in HSV, hue, saturation and value. This will allow us to fine-tune the mask to isolate the desired object. I researched how to split the image:



<http://www.emgu.com/wiki/files/3.2.0/document/html/ff3a9cd6-b726-b9d9-3be9-128a1cba7dbb.htm>

I added this into my code and what happens is you get three new grey scale images that contain one element of H, S and V.



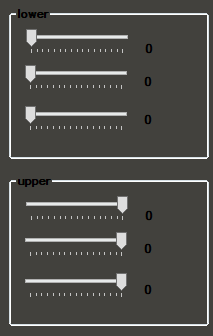


The channels array contains the now split grey scale images for hue saturation and value.

Once I have done this I want to have a threshold value for my colour. You do this by declaring your upper and lower bounds. This is done by creating a different HSV value for the upper and lower bound. Therefore, this will be a range of colours that the program will recognise.



I added six sliders to my form, to control the upper and lower bounds of this mask. Each one controls either H, S, or V.



I made the values of my upper and lower bounds equal to the value of a slide-able bar in my UI so that the values could be changed and monitored in real time. You then declare a new grey scale image. That only accepts the values within the declared range.



This means that all the pixels in the image that fall between the upper and lower bounds are white, and everything else is black. Because there is such a sharp difference between black and white, EMGU is able to detect the outline of the object you isolate.

Then you output that image to a picture box and you can view the mask you have created.

The live preview and sliders make it possible for you to accurately pinpoint the optimal values for the mask and get the best results when it comes to recognition.

Now, using the mask, I want to find the edges of the object I wish to track. I can do this using the FindContours function, which detects edges of lines/colours. Because the mask is just black and white, any contours should be very easy to track.

I research how to use this function and how to draw it onto an image.

<https://www.youtube.com/watch?v=fT9o3F4g3rE>

This video provides a comprehensive tutorial of how the functions work, however I had to input my masked image as opposed to a regular grey mask that the tutorial used:

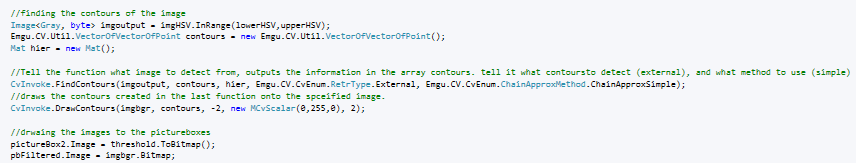


So I instead added:

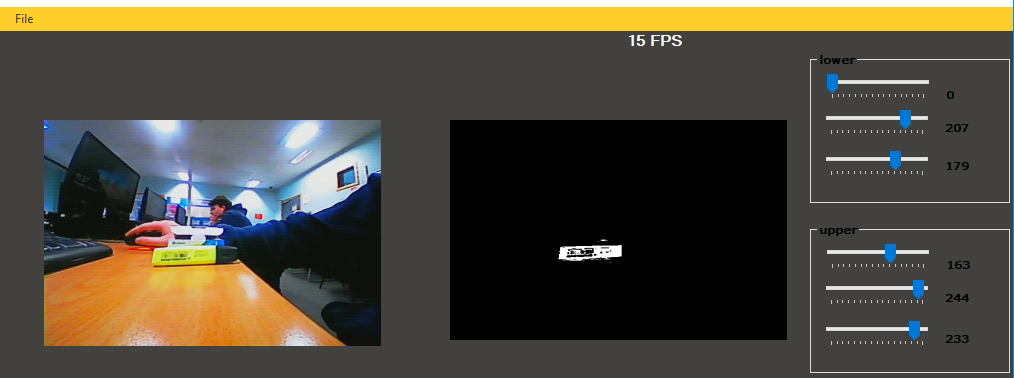


This allows me to find the contours of my own mask set using the sliders. I then output that masked version of the image to a new picturebox. This allows the user to see the changes they are making to isolate the correct colour and also see the regular RGB version in the original preview box

This is what the code for finding and drawing the contours looks like:



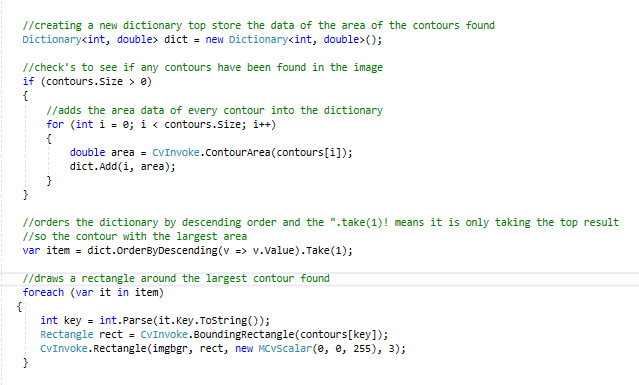
This all together gives me an image that looks like this:



In the future I will make the masked picturebox smaller to declutter the UI this is more for a visual representation of how the mask works

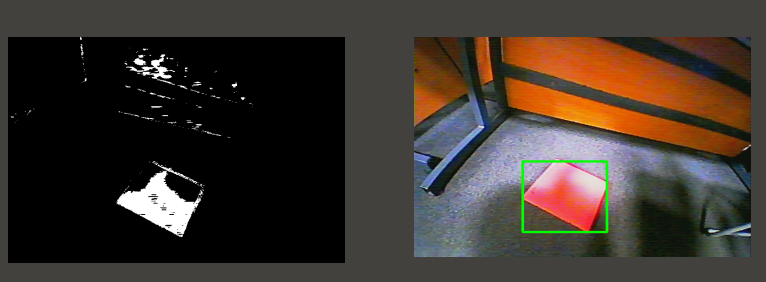
I now want to be able to track the largest contour found on my image. I already know about binding rectangles, you are able to draw a rectangle around your contours. However, I only want to draw around the biggest contour in the image so to do this I have to order my contours by size.

I need to create a dictionary that adds all the contours found, gives them a number and stores their area. I then need to order it from largest to smallest. Then draw a binding rectangle around the biggest one. This is the implementation:



We create a dictionary based on size that stores all of the contours found in the image. Then on the biggest contour found, we draw a bounding rectangle. The “(0, 0, 255)” is the colour of the rectangle. The “3” is the thickness of the rectangle

This gives me an image that looks like this:



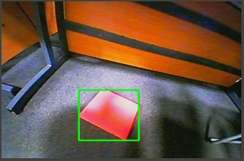
|  |  |  |
| --- | --- | --- |
| Test | Expected outcome | Actual Outcome |
| Changing the sliders when the camera isn’t connected | The slider value label doesn’t change from 0 and the program stays the same | **Pass** |
| Overlapping the upper and lower sliders | The mask doesn’t accept any value from the respective value and the image is black | **Pass** |
| Isolating bright coloured objects. E.g. highlighters, books, cards etc. | Easy to adjust sliders to isolate the desired colour. | **Pass** |
| Isolating duller objects with less contrast in harshly lit environments | Harder to adjust sliders and a less clear object is isolated | There are more false positives. Harder to isolate the object alone, but can be done so that desired object is the biggest white space found on the preview |

**Linking back to success criteria**

Success criteria 3, 4, 5 and 6 are met in this prototype. The user can now isolate and find the contours of the desired object, which will allow us to track the isolated image, the additions to the UI are the sliders, which allow the user to control the mask and select the colour of the object they wish to track. They are easy to use and are accurate. I have also added a second picturebox, to allow the user to see the masked image to isolate the desired object. The time taken to isolate the desired colour is short, usually taking around 15 seconds. We also add to the preview box and see the object being recognised by the program as we paint a bounding rectangle that is a visual representation of what the computer is doing at that time. The bounding rectangle is bold, brightly coloured and clear for the user and class to see.

## Prototype 3

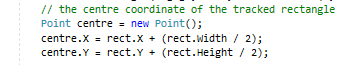
After I have the object tracked, I want to know the X and Y coordinates of the object so I can run an algorithm to get the car to turn to correct its movement. I am splitting the image into a 3x3 grid like this:



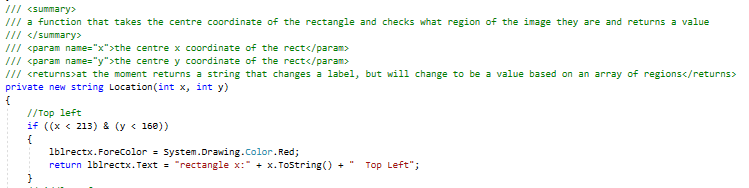
This grid will not actually appear on the screen it will be a virtual grid. In addition, based on which of the nine segments it is in an algorithm runs to turn the car

Splitting the image up into nine segments allows me to have nine different algorithms for car movement run. The logic behind this is that if the object is in the middle centre segment the car will have to accelerate and keep going forwards, as the car travels closer the object will eventually be in the bottom centre segment where the car will stop. This means that the algorithm in each segment’s aim is to position the car such that the object is in the centre of its view and then accelerate until the car is directly in front of the desired object.

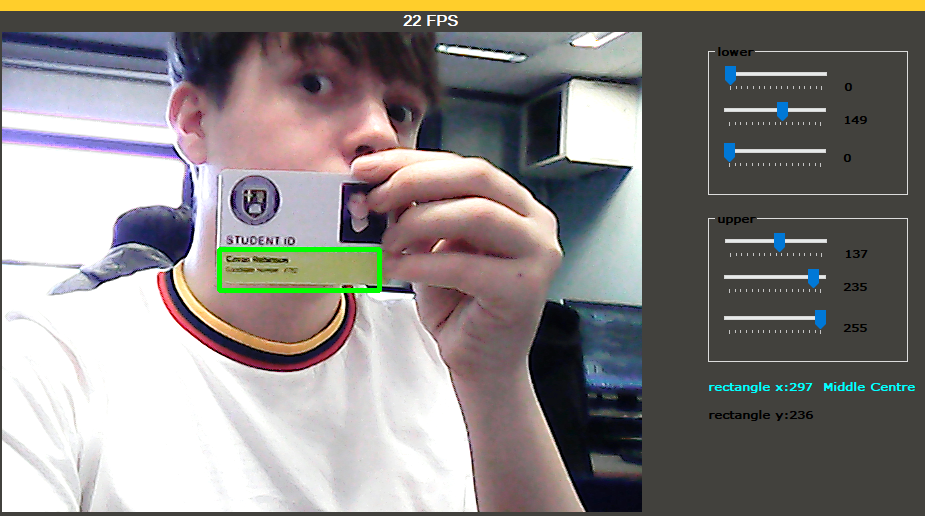
I will do this by finding the centre coordinate of my object, and then a series of if statements determined by the coordinates of the grid decide what happens



rect.X and rect.Y are the coordinates of the top left point of the object rectangle. I create new coordinates for the centre of my object.

I then made the function Location() to run a piece of code based on which segment the centre of the object in in. here is the function:

The grid is constructed entirely in code and is not seen by the user. Each segment is a range of coordinates that pertain to that area of the image. Then if the object’s centre is within that particular segment of the image, the label should display the name of the segment and change to a unique colour for each segment.

Now the label “lblrectx” changes colour and tells you what section of the image the object is in. like this: 

As we can see the location, function runs the code that changes the label depending on the current segment. Next will be to get the car control working and implement the car control code into the location function.

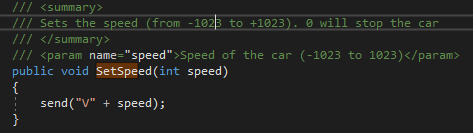
|  |  |  |
| --- | --- | --- |
| Test | Expected outcome | Actual outcome |
| Contents of X and Y labels when no camera is connected | Rectangle X:  Rectangle Y: | **Pass** |
| Contents of label x when object is on the middle centre segment | Rectangle x: middle centre  And the colour changes to blue | **Pass** |
| Contents of label x when object is in any other segment | Rectangle x: (name of current segment)  And the colour changes specific colour | **Pass** |
| Contents of label x when no colour has been isolated | Rectangle x: middle centre  And the colour changes to blue  This is because if no isolation has been done the entire image is marked as a contour | **Pass** |

**Linking back to success criteria**

The grid I have constructed allows us to run specific algorithms based on the objects location, this means we are now able to track the object and name where a bouts it is. And manoeuvre the car based on this information this means that we can control the car with these parameters. The only changes to the UI are to display what section of the screen the object is in. This is a good demonstration of the workings of the code. The end user will be able to see where the object is in the preview and then see how that changes other aspects of the program. This directly links back to my stakeholders requirements as they wanted to clearly see the process that was happening and to interactively show the tracking and computer vision working, this visual feedback is both clear and interactive.

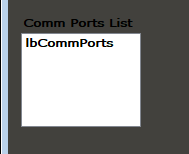
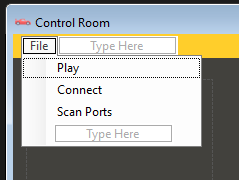
## Prototype 4

Now the code for the camera preview and object detection is done I need to go about connecting the car to the PC. The code that sends and receives signals from the microbit car is pre-made, by the stakeholder. In addition, is written in C#. It works by sending radio signals over the microbit connected to the PC to the microbit in the car telling it what to do. I will create an object of the “CarContoller” class and then you have methods such as “SetSpeed” that send simple commands to the car for example:



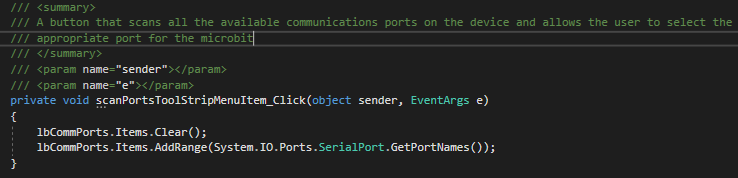
All I need to do is link this class to my program and I should be able to control the car.

I do however need to add to my UI. I need a listbox that displays and allows the user to select from the list of available communications ports on the microbit. For this, I need a “Scan Ports” button that scans and outputs them in the box. In addition, I need a connect button that connects the program to the car.



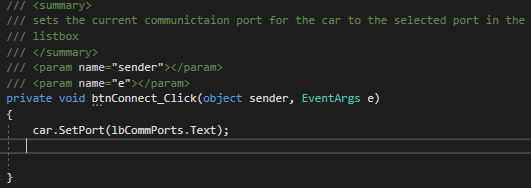
**The code behind this:**

**Scan ports:**



First, the button clears the listbox. Then it adds all of the serial ports available in the device to the listbox.

**Connect:**



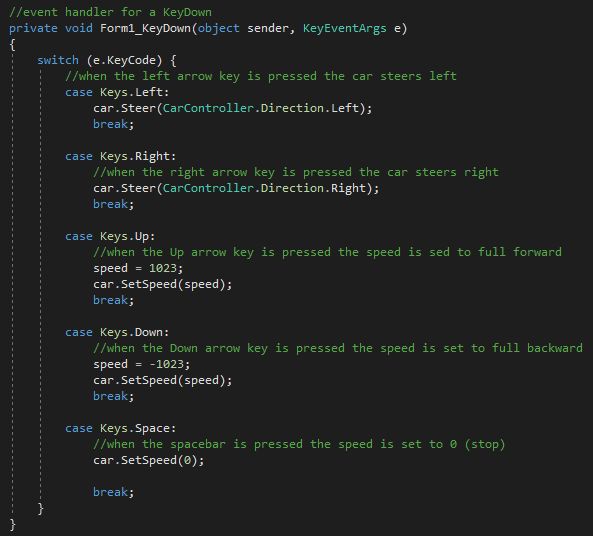
This button sets the communications port for the car to the one currently selected from the listbox.

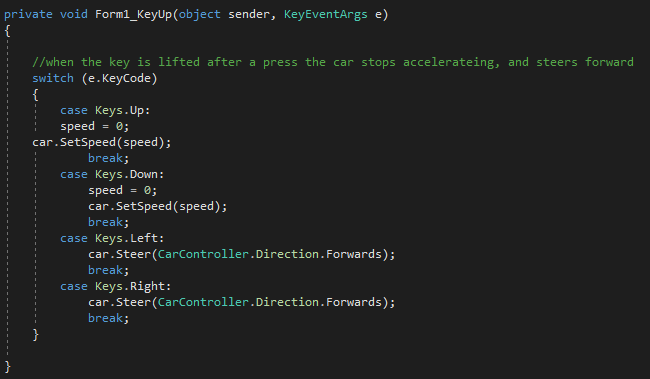
With this done I need to add the manual control over the car in my program.

I make a new object of the “CarController” class called “Car”:



Then, I add controller commands using event handlers:

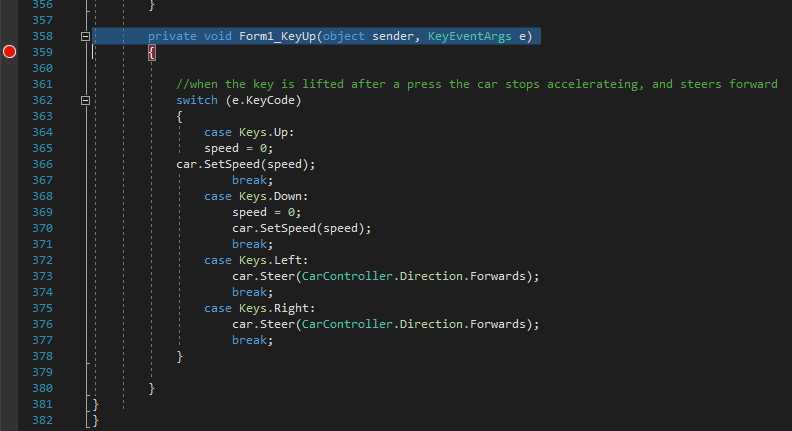




|  |  |  |
| --- | --- | --- |
| test | Expected outcome | Actual outcome |
| Hold down the up arrow key | The car accelerates | **Pass** the car accelerates |
| Lifting the up arrow key | The car stops accelerating | **Fail** the car continues to accelerate |
| Hold down the down arrow key | The car reverses | **Pass** the car reverses |
| Lifting up the down arrow key | The car stops reversing | **Fail** the car continues to reverse |
| Pushing the right and left arrow keys | The car steers left and right respectively | **Pass** the car steers left and right respectively |
| Lifting up the right and left arrow keys | The car returns to a forward position | **Fail**  the car stays in the direction |

### Addressing the failed tests

All of the tests involving lifting the key are fails. This means there is something wrong with the vent handler. It must be a logic error because I am not seeing any errors in my list.



I put a breakpoint on the line of the KeyUp event handler. Then I ran the code, but when I lifted the arrow keys the breakpoint didn’t activate, meaning the cod wasn’t even running.

There was nothing different. I checked the designer and found that there was no KeyUp event compiling



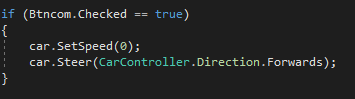
Therefore, I added an event handler underneath the KeyDown handler.

|  |  |  |
| --- | --- | --- |
| test | Expected outcome | Actual outcome |
| Lifting the up arrow key | The car stops accelerating | **Pass** the car stops |
| Lifting up the down arrow key | The car stops reversing | **Pass** the car stops |
| Lifting up the right and left arrow keys | The car returns to a forward position | **Pass**  the car returns to the forwards direction |

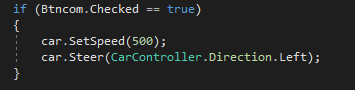
## Prototype 5

Now I will begin coding the self-controlling part. The theory behind the control is that based on the segment the car will do a different thing:

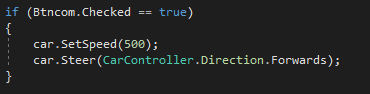
**Bottom three sections –** the car stops accelerating, if the object is in the bottom left the car stops and steers left, the bottom centre, the car stops and steers forwards, and if the object is in the bottom right, the car stops and steers left.



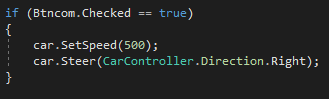
**Middle left –** the car accelerates at 500 out of 1023 and steers left



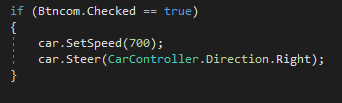
**Middle centre –** the car accelerates at 500 and steers forwards



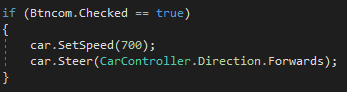
**Middle Right –** the car accelerates at 500 and steers right



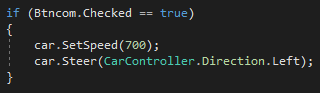
**Top Right –** the car accelerates at 800 and steers right



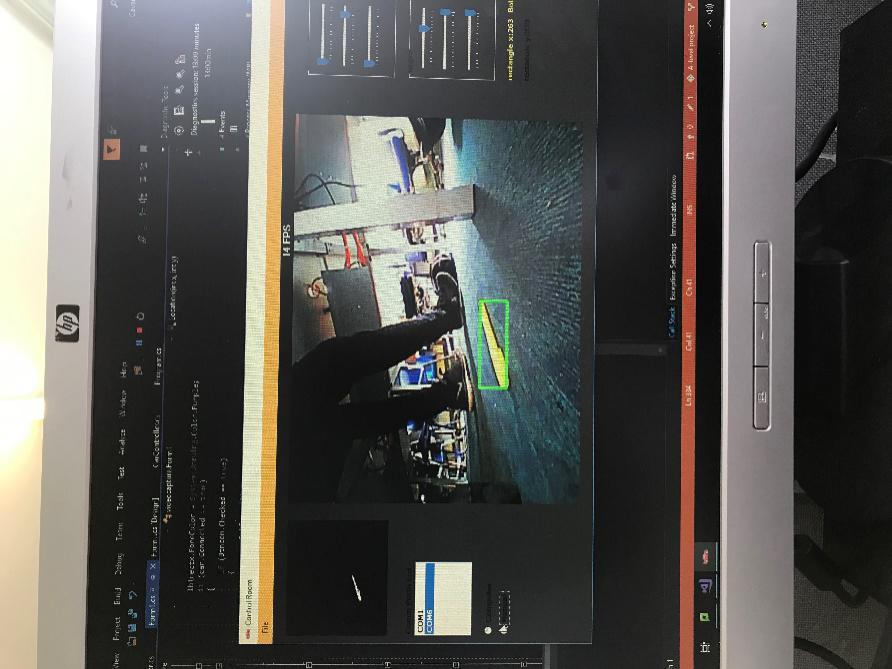
**Top Centre –** the car accelerates at 800 and steers forwards



**Top Left -**  the car accelerates at 800 and steers left



**It works:**





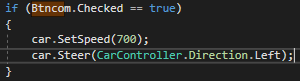
|  |  |  |
| --- | --- | --- |
| test | Expected outcome | Actual outcome |
| Placing the object in the bottom section of the car’s view | The car stops accelerating | **Pass** the car stops |
| Placing the object in the middle centre | The car accelerates and steers forward until the object appears in the bottom section | **Pass** |
| Placing the object in the top centre | The car accelerates at a faster speed and steers forwards until the object appears in the middle centre | **Pass** |
| Placing the object in the left sections | The car turns to the left and accelerates until the object is in one of the centre section | **Pass** |
| Placing the object in the right sections | The car turns right and accelerates until the object appears in one of the middle sections | **Pass** |

However, there were a few issues. As soon as I began debugging and connected the camera and car, the car would immediately begin driving toward an object before the user had a chance to calibrate or set up the test environment.

To fix this I added the radio buttons from my desired UI:



Upon start-up, the manual box is checked automatically. The computer controlled movement now only works if the computer controlled button is checked.



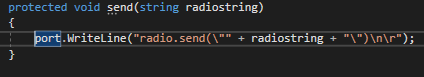
This means that the user can safely and reliably switch between manual and computer controlled mode. This implementation both fits functionality and the design specified by the stakeholder.

|  |  |  |
| --- | --- | --- |
| Test | Expected outcome | Actual outcome |
| Controlling the car with the arrow keys with manual checked | The car should control as expected without the computer control active | **pass** |
| Trying to control the car with the arrow keys when computer control is checked | The arrow keys should not control the car | **pass** |
| Checking the computer control button in the UI | The car should switch to computer control and start moving by itself | **pass** |
| Connecting the camera with no car connected | The camera should connect as expected and the car control should only run when the car is connected | **Fail**  the program crashes fatally causing me to have to log out of my account and back in again |

### Addressing the failed tests

The failed test is important, as it could be very damaging to the user if the crash, not only renders the program unusable but the rest of the system as well.

I get an error message that tells me there are no ports connected and it links to a line in the “CarController” class.



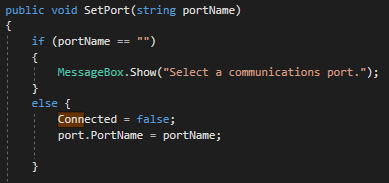
This line initiates the communication between the two microbit. The code is breaking when the program attempts to connect the car that isn’t there or if no comport has been selected.

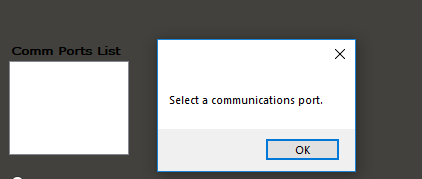
This means that the code for the camera is now trying to connect to the car. So the camera code will not work if the car is not connected.

To fix this I add a Boolean

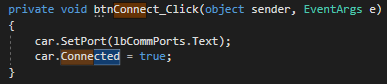


I then show a messagebox if the user doesn’t select a comport saying.

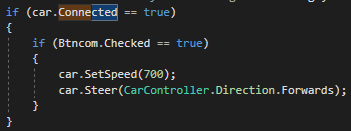




Then when the user scans and selects a commPort in the button “connect” code the connected Boolean is set to true



Then I add an if statement to my entire computer controlled car movement based on the connected variable:



|  |  |  |
| --- | --- | --- |
| Test | Expected outcome | Actual outcome |
| Connecting the camera with no car connected | The camera should connect as expected and the car control should only run when the car is connected | **Pass** |
| Attempting to connect to the car without selecting a comport | Messagebox telling the user to select a port | **Pass** |

**Linking back to the success criteria**

This iteration adds the computer controlled aspect of the success criteria. It also adds the simple buttons to both initiate and interrupt navigation. The follow function, turned out to be redundant as the car already continues to follow the object until the manual control is checked. The use of radio buttons helps with validation, as it is impossible for more than one to be checked at once.

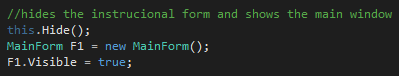
## Prototype 6

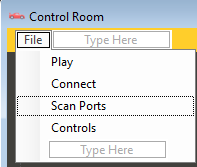
**Adding the instructional window**

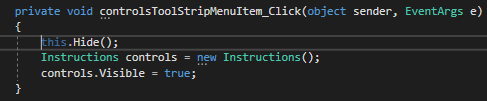
With the main form finished, I need to add my instructional widow. I do this by going into my project and adding a new windows form into the project. I then call this form, “Instructions”

I change the start-up form to the instructional form and start adding my instructions:

I just added a series of labels including a welcome to the demonstration and how to use the program. They will educate the user on how to set up the program, to navigate to the object that they want, and how to control the car itself. I also added a button to close the instruction window and open the main form this is what the code for the button looks like:

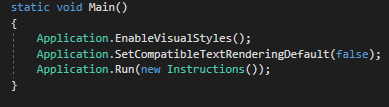
 This.hide() hides the instructional form, and we create a new object of MainForm class and make it visible.

 I also added a controls button that hides the main form and opens the instructional form again, in case the user forgets or wants to go over the controls. The code for that button is as follows.



The code is mostly the same only makes a new object of the instruction class instead of form 1. This means the user can now seamlessly switch between the two forms and check the controls when needed.

I want to have the instructional form to be the window that opens when you start the program, so I need to change the initial start form. Therefore, I go into my program.cs code and change my run application:

 The instructions part means that the instructions form will be the window that opens first.

# Evaluation

## Post Development testing

I carried out a series of tests when my implementation was complete to ensure

|  |  |  |
| --- | --- | --- |
| test | Expected outcome | Actual outcome |
| Pressing any keys in the instructional window | No effect does not crash the program | **Pass** |
| Pressing the “Got it!” Button in the instructional window | The instructional window close and the main window opens | **Pass** |
| Pressing the scan ports button | Any available ports should appear in the listbox | **Pass** |
| Pressing the connect button, after selecting a port | The car connects to the program and is controllable | **Pass** |
| Pressing the connect button without selecting a port | A message box tells the user to select a communications port | **Pass** |
| Pressing the up arrow | The car accelerates | **Pass** |
| Lifting the up arrow | The car stops accelerating | **Pass** |
| Pressing the down arrow | The car reverses | **Pass** |
| Lift the down arrow | The car stops reversing | **Pass** |
| Press the left arrow | The car steers left | **Pass** |
| Lift the left arrow | The car returns to the forward position | **Pass** |
| Press the right arrow | The car steers rights | **Pass** |
| Lift the right arrow | The car returns to the forward position | **Pass** |
| Pressing the play button | The camera preview starts and displays an FPS timer | **Pass** |
| Using the arrow keys in manual mode | The arrow keys should control the car | **Pass** |
| Moving the slider | Should change the colour values of the mask to isolate a colour | **Pass** |
| Overlapping upper and lower sliders | The mask is fully black as no colour values are accepted and so no object can be found | **Pass** |
| Pressing the controls button | The instruction window opens and the main form is hidden | **Pass** |
| Pressing the file button | The tool strip drops down revealing the other buttons | **Pass** |
| Try and connect when no camera is connected | Messagebox telling the user to connect the camera | **Pass** |
| Place similar looking items in front of the car during navigation | Ignore the similar item | Intermittently in non-harsh lighting, the program is able to discern between two similar looking objects, however in harsh lighting two colours can be mistaken and the car may begin to follow this object. |
| click the computer button | The car locks the user out of manual control and begins being controlled by algorithms alone | **Pass** |
| To click the computer button with the camera not connected | The car remains in place until a camera is connected | **Pass** |
| To click navigate without any objects found | The car stays in place until an object is located | **Pass** |

**Stake holder black box testing:**

I had my stakeholders black box test the product to see if it met the requirements we had agreed on and to see how the usability holds up.

**Mr. Dring**

|  |  |
| --- | --- |
| Requirement | Met? |
| Intuitively designed UI | **Pass** Yes the UI was easy to understand and navigate |
| Clearly demonstrated video feed | **Pass** The video feed was large, clear and interesting and gave a good view of what the camera is seeing and what the computer is detecting. The masked video feed means that I can adjust the sliders to track the object I want. |
| Manual control | **Pass** I just connect, and start using the arrow keys to control the car, this worked as I had imagined. |
| Clear way of selecting a tracking colour | **Pass** the sliders on the right mean that I can in real time pick what colour I want to track and the masked video means that I can see what I’m doing and make adjustments |
| Algorithms that navigated to the desired object | **Pass** after I’ve connected and isolated the object I want to navigate to or follow I just click the computer button and it goes |
| Follow function | Sort of, there is no dedicated follow function; the car will automatically keep following the object until the manual mode is reselected. |
| A wireless FPV camera | **Pass** a small fpv camera on the front of the car does the job well |
| Car control program | **Pass** the car control program works as desired |
| Instructions page | **Pass** the instructions page taught me what I needed to do before I got into the main window. Meaning I knew what I was doing when connecting the car and camera, and setting up the object detection. |

Me: “Did you run into any crashes or bugs?”

Mr. Dring: “no, the program was robust and if I tried to do something that could potentially break the program I get messages telling me what to do”

**Harry Brown**

|  |  |
| --- | --- |
| Requirement | Met? |
| Interactive for students | **Pass** the instruction page tells me that the demo is interactive and teaches me the controls. The manual controls are easy to use so you can do it yourself |
| Manual control | **Pass** the arrow keys are a good way to control the car. They are responsive and fast. |
| Able to pick the things I track | **Pass** I can use the sliders to pick the objet and I can use the little box in the top left to see what I’m doing |
| The car should drive to the object I want | **Pass** once I click computer the car drives to the object I have picked, using the slider. |
| I should be able to see the camera feed and have a box around the object | **Pass** The box is clear and brightly coloured so I can see what the object is tracking |

Me: “did you run into any crashes or bugs?”

Harry Brown: “no, I did everything the instructions said, so nothing went wrong. Everything worked as it was meant to”

I also had a student, Yaro Shiroki, unfamiliar with the project test the program by going through it, as a normal demonstration would take place this was what he found:

“The UI is simple and easy to understand and use”

“The only button I would find hard to understand without the instructions is the scan ports button”

“With no help I was able to successfully connect to the car, and the camera. Control the car myself, and set the car to follow the object I wanted.”

“I tried to crash the program and I couldn’t manage it!”

“I really like the fact you can see the camera feed in the program, I can control the car without even being able to see it”

“The preview means that I can see what the computer is tracking and the little labels on the right show that the program knows what part of the screen it is in”

## Success criteria

**1. Car should be manually controllable using the arrow keys**

The solution **successfully** meets these criteria; in [prototype 4](#_Prototype_4), I added the car controls. The car uses two microbits that communicate with one another. When the user presses one of the arrow keys, code pertaining to a drive state of the car runs and sends a radio signal to the microbit on the car.

**2. Large and clear model**

I asked if the testers whether or not the car was big enough:

Me: “did you think the car was big enough to see in a classroom setting and do you think an entire class could see?”

Mr. Dring: “yes I think the car was a good size and would be large enough for a whole class”

Harry Brown: “yeah I think the car is a good size”

Based on the user’s feedback this criterion was met **successfully.**

**3. Fast and responsive**

The manual controls are responsive and harry brown said “the controls feel responsive and fast.” Set up takes less than a minute, and isolating the object takes less than 20 seconds. I asked harry brown and yaro shiroki if they thought the connection was fast enough:

Yaro: “the when you actually press connect to the car, it is almost instant. The camera takes like 5 seconds or so and setting up is pretty fast.”

Harry: “the whole thing can be set-up, connected and driving in less than a minute, take another half a min for getting the object tracking and the car can drive itself. I think it’s fast.”

Based on the user’s feedback this criterion was met **successfully.**

**4. A feed that sees what the computer sees**

This criterion was **successfully** met in [prototype 1](#_Prototype_1), and [2](#_Prototype_2). I used Emgu to start a videocapture and pant each frame to a picturebox. I then added a second picturebox that shows the feed with the mask overlaid, which allows the user to see which colours they are isolating. After this, I added a bounding rectangle onto of the image showing the object that the user is tracking. When I added the grid. In prototype three, I also added labels that show what section of the 3x3 grid the object is in.

**5. Intuitive and clear UI**

This success criterion was to have a student with no experience of the program should be able to operate the program without much prompting. When Yaro tested the program, I asked him what he thought of the UI and asked him if he could operate the program without much prompting.

“I managed to use all the features of the program without any prompting, the instructions at the start meant that I knew what I had to do before I went in and if I forgot anything I could go back and check the controls whenever. Stuff was labelled well so I could understand it.” This criterion was met **successfully**.

**6. The selection of a tracking colour should be simple**

This criterion was **successfully** met in [prototype 2](#_Prototype_2). I added six sliders, which control the upper, and lower bounds of the HSV mask. Then I display the video feed with the mask in a small picturebox in the top left of the form. This allows the user to see what they are isolating when they change the colour values. The masked box is black and white, whatever is in white is being tracked, and the black is excluded. The biggest contour found in the masked image has the rectangle drawn around it in the main form, so the user can verify if their desired object is surrounded by a green rectangle.

**7. Simple button that initiates navigation**

This criterion was **successfully** met in [prototype 5](#_Prototype_5) when I added the radio buttons “computer” and “manual” the computer button initiates the navigation, the car will drive to the centre of whatever object is being tracked. By default, the manual button is selected and manual and computer modes cannot both be activated at the same time.

**8. Instructional window**

This criterion was **successfully** met in [prototype 6](#_Prototype_6). I added another form to the project containing a brief on what the demonstration is, and a control scheme. Then a button closes the window and starts the main window. The instructional window teaches the user how to use the program before they get into the main form. A student or teacher could run the program for the first time and be able to understand the whole thing, because of the instructions. If they forget the controls, they can simply press the controls button and see them again.

**9. The ability to interrupt the navigation**

This criterion was **successfully** met in [prototype 5](#_Prototype_5); the buttons “computer” and “manual” are radiobuttons meaning only one of them can be checked at once, if you wish to interrupt the navigation, you simply press the manual button and the computer control stops; this is good as it also mean the car cannot be controlled manually in computer-controlled mode.

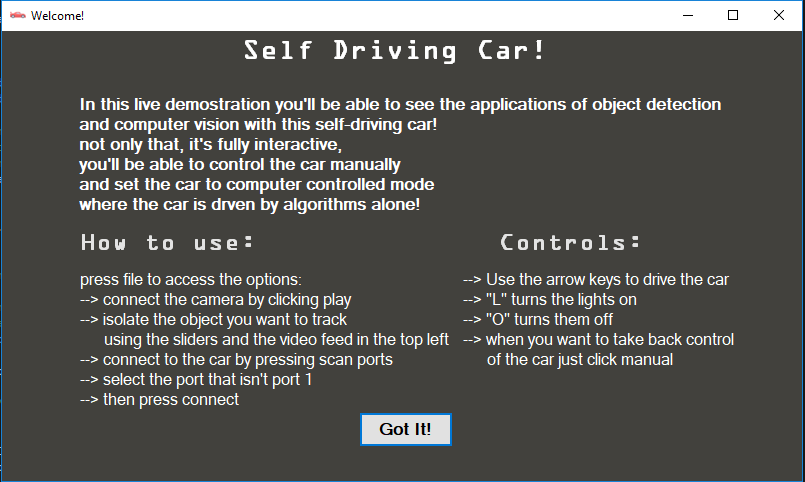
**10. A function that follows a moving object**

There is no dedicated follow function as the code I made for navigation is already able to follow a moving object, a separate function turned out to be redundant, so this criterion is **partially met** in [prototype 5](#_Prototype_5). In this prototype, we add turning algorithms to each of the segments of the 3x3 grid made in code. Depending on which segment of the grid the object is in, a separate control algorithm will run to turn the car to face the object and drive toward it. This process continues indefinitely until the manual function is selected. This means that the car will already follow a moving object if the user keeps the computer-control mode active.

## Usability Features

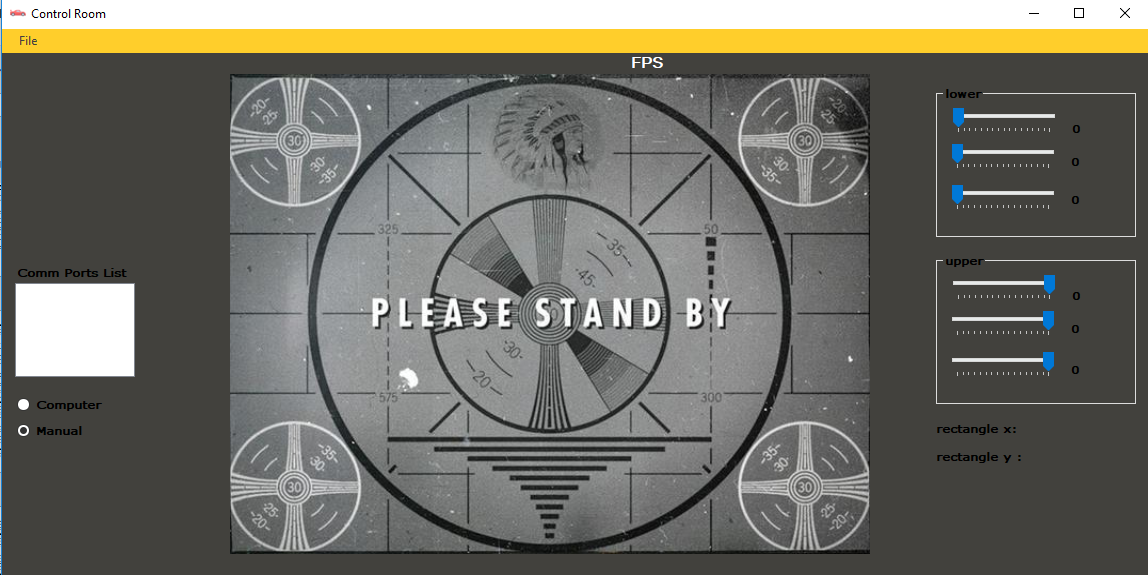
The features that aid the usability of my project are:

**Instrucitonal window**



The instructional window aids with usability because it educates the user before they get into the main form, this means they are less confused by the methods of connection or calibration, and are less likely to give invalid inputs.

**Few buttons in the main form**

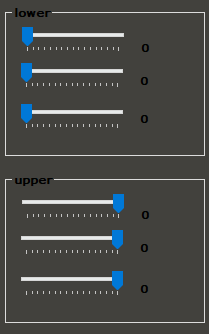


The main form is not cluttered with many buttons. All main function buttons are in the toolstrip, accessed by pressing the “file” button. This makes the main form clearer and enables me to make the preview window as big as possible so a class can view it from a distance.

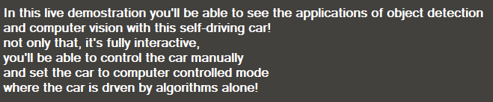
**Simple control scheme**

The control scheme for the car is the arrow keys, which are intuitive and easy to use, so anyone can pick it up easily, and the method to activate navigation is one button

**Method of colour picking**

The sliders allow the user to adjust the HSV mask in real-time meaning it is easier to isolate the colour of your object and the masked picturebox allows the user to see the changes they are making. The sliders are intuitive and accurate, the label to the right allows the user to see the exact HSV values of their upper and lower bounds.

**Clear and easy to read font and colour scheme**



I use a simple, legible white font against a dark grey background. The toolstrip is yellow, and the font on there is black.

I think that I successfully met all of the usability features that I wanted and that were required in my success and stakeholder requirements.

## Future Development

The partially met success criteria 10 (to have a function that follows a moving object) can be solved in future development. What I would do is adapt the current computer controlled code to stop navigating once the car gets within a certain proximity to the object, when the object meets one of the bottom sections of the 3x3 grid. Then if the follow function is active if the object reaches the bottom, section the code continues to run until the manual button is pressed. This would require a follow button to be added to the UI. Essentially the follow function would be the same as the current computer controlled mode.

The future development of the follow function would change the usability of the program. The computer controlled and follow buttons should be in their own group box. The follow button should be a checkbox that can only be activated when in computer-controlled mode. It would have to be made clear in the instructional window as well.

**Things that I would add in the future**

Some features I would add in the future would be a search function in the computer-controlled mode, if the object you want to track were not within the cars current view you could activate a search function in which the car would travel in a radius around the room until it finds the object. This would mean the user could do even less and even more of the cars function can be controlled by the program itself. This would be another example and demonstration of computer vision.

## Limitations

Now the computer-control only works if the object is in the camera’s field of view. This means the user has to drive the car until the object is within the cars FOV. The search function talked about in the future development section would rectify this limitation

The comports listbox currently lists two ports when scanned, commport1 which is a communications port on the computer, and the available communications port on the microbit. A limitation is that the commport 1 will never be used so this listbox is essentially redundant and the commport could be chosen in code without the user having to do it themselves. This would drastically improve usability, as currently the need to scan and select ports before connection is the most confusing part of the design. To fix this I would add code that selects any communications port that is not port one. This would fix the issue, as the other available port is always the port on the microbit.

The lighting of the environment the car is in can greatly hinder the effectiveness of the object detection function. If the lighting is particularly harsh, colours can appear “washed out” and are harder to define in the mask. To fix this I would have the mask adjust for differences in lighting and perhaps upgrade the camera to one with a higher range of colour, meaning that it is easier to discern between two possible colours.

# Appendix

## MainForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

using System.Management;

using Emgu.CV;

using Emgu.CV.Structure;

using Emgu.CV.Util;

namespace videocapture

{

//the main class for the image processing, includes the videocapture, the FPS counter and the find contours code.

public partial class MainForm : Form

{

//creates a new object of the car controller API

protected CarController car = new CarController();

//declares a new videocapture called capture and declares some of the global variables

VideoCapture capture;

bool Pause = false;

int framecount = 0;

bool cap = false;

int speed;

public MainForm()

{

InitializeComponent();

//when the form loads the default values for the mask are declared

tbhup.Value = 255;

tbsup.Value = 255;

tbvup.Value = 255;

tbhlow.Value = 0;

tbslow.Value = 0;

tbvlow.Value = 0;

}

/// <summary>

/// when the play button is pressed all of the image proccessing is done in this function, the video feed is displayed in the form and the object is located

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

public async void playToolStripMenuItem\_Click(object sender, EventArgs e)

{

//runs the camera connect function to see if the camera is connected

bool camera\_check = Camera\_Connect();

//clears the picturebox for the preview

PBPreview.Image = null;

if (camera\_check == true)

{

Pause = false;

try

{

//the fps timer starts when the play buuton is pressed

TimerFPS.Start();

capture = new VideoCapture(0);

while (!Pause)

{

Mat m = new Mat();

//the image stored in the matrix "m" (which is the current frame of the video) is now stored as a single image that is input inot the picturebox

capture.Read(m);

//value of the framecount increments by 1 everyt time a new images s inputted into the picturebox

framecount += 1;

if (!m.IsEmpty)

{

//declares two images onw HSV for the mask and one BGR for the user to view in the proper colour space

Image<Hsv, Byte> imgHSV = m.ToImage<Hsv, Byte>();

Image<Bgr, byte> imgbgr = m.ToImage<Bgr, byte>();

//splits the HSV image up into three, single channeled, grey scaled images.

//sets the threshold values to equal the values on the sliders in the UI

Image<Gray, Byte>[] channels = imgHSV.Split();

Hsv upperHSV = new Hsv(tbhup.Value, tbsup.Value, tbvup.Value);

Hsv lowerHSV = new Hsv(tbhlow.Value, tbslow.Value, tbvlow.Value);

//outputs the values of the upper and lower bound values to the labels next to the sliders

Lbllowh.Text = tbhlow.Value.ToString();

Lbllows.Text = tbslow.Value.ToString();

Lbllowv.Text = tbvlow.Value.ToString();

Lbluph.Text = tbhup.Value.ToString();

Lblups.Text = tbsup.Value.ToString();

Lblupv.Text = tbvup.Value.ToString();

//setting the image to include the mask threshold set using the sliders.

Image<Gray, Byte> threshold = imgHSV.InRange(lowerHSV, upperHSV);

//finding the contours of the image

Image<Gray, byte> imgoutput = imgHSV.InRange(lowerHSV, upperHSV);

Emgu.CV.Util.VectorOfVectorOfPoint contours = new Emgu.CV.Util.VectorOfVectorOfPoint();

Mat hier = new Mat();

//Tell the function what image to detect from, outputs the information in the array contours. tell it what contoursto detect (external), and what method to use (simple)

CvInvoke.FindContours(imgoutput, contours, hier, Emgu.CV.CvEnum.RetrType.External, Emgu.CV.CvEnum.ChainApproxMethod.ChainApproxSimple);

//draws the contours created in the last function onto the spceified image.

//creating a new dictionary top store the data of the area of the contours found

Dictionary<int, double> dict = new Dictionary<int, double>();

//check's to see if any contours have been found in the image

if (contours.Size > 0)

{

//adds the area data of every contour into the dictionary

for (int i = 0; i < contours.Size; i++)

{

double area = CvInvoke.ContourArea(contours[i]);

dict.Add(i, area);

}

}

//orders the dictionary by descending order and the ".take(1)! means it is only taking the top result

//so the contour with the largest area

var item = dict.OrderByDescending(v => v.Value).Take(1);

foreach (var it in item)

{

int key = int.Parse(it.Key.ToString());

//draws a rectangle around the largest contour found

Rectangle rect = CvInvoke.BoundingRectangle(contours[key]);

CvInvoke.Rectangle(imgbgr, rect, new MCvScalar(0, 255, 0), 3);

// the centre coordinate of the tracked rectangle

Point centre = new Point();

centre.X = rect.X + (rect.Width / 2);

centre.Y = rect.Y + (rect.Height / 2);

lblrectx.Text = "rectangle x:" + centre.X.ToString();

lblrecty.Text = "rectangle y:" + centre.Y.ToString();

Location(centre.X, centre.Y);

}

//drwaing the images to the pictureboxes

PBPreview.Image = imgbgr.Bitmap;

pbFiltered.Image = threshold.ToBitmap();

//retrieving the fps property from the capture properties feature, and then setting the delay to the FPS value

double fps = capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Fps);

await Task.Delay(Convert.ToInt32(fps));

}

else

{

break;

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

else

{

MessageBox.Show("Connect the camera");

}

}

/// <summary>

/// the timer that ticks every second pasting the count of the frames to a label

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void TimerFPS\_Tick(object sender, EventArgs e)

{

double fps = framecount;

lblFPS.Text = fps.ToString() + " FPS";

framecount = 0;

}

/// <summary>

/// sets the current communictaion port for the car to the selected port in the

/// listbox

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnConnect\_Click(object sender, EventArgs e)

{

car.SetPort(lbCommPorts.Text);

car.Connected = true;

}

/// <summary>

/// a function that takes the centre coordinate of the rectangle and checks what region of the image they are and returns a value

/// </summary>

/// <param name="x">the centre x coordinate of the rect</param>

/// <param name="y">the centre y coordinate of the rect</param>

/// <returns>at the moment returns a string that changes a label, but will change to be a value based on an array of regions</returns>

private new string Location(int x, int y)

{

//Top left

if ((x < 213) & (y < 160))

{

lblrectx.ForeColor = System.Drawing.Color.Red;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(700);

car.Steer(CarController.Direction.Left);

}

}

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Top Left";

}

//Middle Left

else if (((x < 213) & ((y > 160) & (y < 320))))

{

lblrectx.ForeColor = System.Drawing.Color.White;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(500);

car.Steer(CarController.Direction.Left);

}

}

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Middle Left";

}

//bottom Left

else if ((x < 213) & ((y > 320) & (y < 480)))

{

lblrectx.ForeColor = System.Drawing.Color.Cyan;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(0);

car.Steer(CarController.Direction.Left);

}

}

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Bottom Left";

}

//Top Centre

else if (((213 < x) & (x < 427)) & (y < 160))

{

lblrectx.ForeColor = System.Drawing.Color.Orange;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(700);

car.Steer(CarController.Direction.Forwards);

}

}

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Top Centre";

}

//Middle Centre

else if (((213 < x) & (x < 427)) & ((y > 160) & (y < 320)))

{

lblrectx.ForeColor = System.Drawing.Color.Aqua;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(500);

car.Steer(CarController.Direction.Forwards);

}

}

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Middle Centre";

}

//Bottom Centre

else if (((213 < x) & (x < 427)) & (y < 480))

{

lblrectx.ForeColor = System.Drawing.Color.Yellow;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(0);

car.Steer(CarController.Direction.Forwards);

}

}

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Bottom Centre";

}

//Top Right

else if (((427 < x) & (x < 640)) & (y < 160))

{

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(700);

car.Steer(CarController.Direction.Right);

}

}

lblrectx.ForeColor = System.Drawing.Color.Green;

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Top Right";

}

//Middle Right

else if (((427 < x) & (x < 640)) & ((y > 160) & (y < 320)))

{

lblrectx.ForeColor = System.Drawing.Color.PeachPuff;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(500);

car.Steer(CarController.Direction.Right);

}

}

//changing the label to indicate object location

return lblrectx.Text = "rectangle x:" + x.ToString() + " Middle Right";

}

//Bottom Right

else if (((427 < x) & (x < 640)) & (y < 480))

{

//driving the car towards the object

lblrectx.ForeColor = System.Drawing.Color.Purple;

if (car.Connected == true)

{

if (Btncom.Checked == true)

{

car.SetSpeed(0);

car.Steer(CarController.Direction.Right);

}

}

return lblrectx.Text = "rectangle x:" + x.ToString() + " Bottom Right";

}

else

{

//changing the label to indicate object location

lblrectx.ForeColor = System.Drawing.Color.Black;

return lblrectx.Text = "rectangle x:" + x.ToString();

}

}

private void Form1\_Load(object sender, EventArgs e)

{

//upon loading the form the manual button is automatically checked

Btnmanual.Checked = true;

}

private bool Camera\_Connect()

{

bool connected = false;

var UsbDevices = USBDeviceInfo.GetUSBDevices();

string report = "";

int i = 0;

foreach (var usbDevice in UsbDevices)

{

if (usbDevice.DeviceID.ToLower().Contains(@"usb\vid\_18ec&pid\_5850"))

{

report += "DeviceID:" + usbDevice.DeviceID + "\n";

connected = true;

}

}

// MessageBox.Show(report);

return connected;

}

/// <summary>

/// A button that scans all the available communications ports on the device and allows the user to select the

/// appropriate port for the microbit

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void scanPortsToolStripMenuItem\_Click(object sender, EventArgs e)

{

lbCommPorts.Items.Clear();

lbCommPorts.Items.AddRange(System.IO.Ports.SerialPort.GetPortNames());

}

//event handler for a KeyDown

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

bool lightState = false;

if (car.Connected == true)

{

switch (e.KeyCode)

{

//when the left arrow key is pressed the car steers left

case Keys.Left:

car.Steer(CarController.Direction.Left);

break;

case Keys.Right:

//when the right arrow key is pressed the car steers right

car.Steer(CarController.Direction.Right);

break;

case Keys.Up:

//when the Up arrow key is pressed the speed is sed to full forward

speed = 1000;

car.SetSpeed(speed);

break;

case Keys.Down:

//when the Down arrow key is pressed the speed is set to full backward

speed = -800;

car.SetSpeed(speed);

break;

case Keys.Space:

//when the spacebar is pressed the speed is set to 0 (stop)

car.SetSpeed(0);

break;

case Keys.X:

//when the spacebar is pressed the speed is set to 0 (stop)

//car.Steer(CarController.Direction.Foxxxx

car.Steer(CarController.Direction.Forwards);

break;

case Keys.D:

car.OpenDoors(true);

break;

case Keys.L:

//if (lightState == false)

// {

car.SetHeadLights(true);

car.SetBrakeLights(true);

lightState = true;

// }

// else

//{

//car.SetHeadLights(false);

// car.SetBrakeLights(false);

// lightState = false;

// }

break;

case Keys.O:

car.SetHeadLights(false);

car.SetBrakeLights(false);

lightState = false;

break;

}

}

}

private void Form1\_KeyUp(object sender, KeyEventArgs e)

{

//when the key is lifted after a press the car stops accelerateing, and steers forward

switch (e.KeyCode)

{

case Keys.Up:

speed = 0;

car.SetSpeed(speed);

break;

case Keys.Down:

speed = 0;

car.SetSpeed(speed);

break;

case Keys.Left:

car.Steer(CarController.Direction.Forwards);

break;

case Keys.Right:

car.Steer(CarController.Direction.Forwards);

break;

}

}

private void controlsToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Hide();

Instructions controls = new Instructions();

controls.Visible = true;

}

}

}

/// <summary>

/// Generates a list of all the USB devices connected to the PC, and information about it

/// </summary>

public class USBDeviceInfo

{

public USBDeviceInfo(string deviceID, string pnpDeviceID, string description)

{

//retrieves the deviceID, pnpDeviceID and a description of the USBDevice

this.DeviceID = deviceID;

this.PnpDeviceID = pnpDeviceID;

this.Description = description;

}

public string DeviceID { get; private set; }

public string PnpDeviceID { get; private set; }

public string Description { get; private set; }

//generating the list

public static List<USBDeviceInfo> GetUSBDevices()

{

List<USBDeviceInfo> devices = new List<USBDeviceInfo>();

ManagementObjectCollection collection;

using (var searcher = new ManagementObjectSearcher(@"Select \* From Win32\_USBHub"))

collection = searcher.Get();

foreach (var device in collection)

{

//gets different properties of the USB device

devices.Add(new USBDeviceInfo(

(string)device.GetPropertyValue("DeviceID"),

(string)device.GetPropertyValue("PNPDeviceID"),

(string)device.GetPropertyValue("Description")

));

}

collection.Dispose();

return devices;

}

}

## Instruction.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace videocapture

{

public partial class Instructions : Form

{

public Instructions()

{

InitializeComponent();

}

private void Instructions\_Load(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

//hides the instrucional form and shows the main window

this.Hide();

MainForm F1 = new MainForm();

F1.Visible = true;

}

}

}

## CarController.CS

using System;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace videocapture

{

public class CarController

{

protected bool \_connected = false;

protected System.IO.Ports.SerialPort port;

/// <summary>

/// Create a new car controller object.

/// </summary>

public CarController()

{

port = new System.IO.Ports.SerialPort();

port.BaudRate = 115200;

}

/// <summary>

/// Sets the COM port the micro:bit is connected to. Use System.IO.Ports.SerialPort.GetPortNames to get a list of available ports

/// </summary>

/// <param name="portName"></param>

public void SetPort(string portName)

{

if (portName == "")

{

MessageBox.Show("Select a communications port.");

}

else {

Connected = false;

port.PortName = portName;

}

}

/// <summary>

/// Connect or disconnects to/from the car

/// </summary>

public bool Connected

{

get

{

return port.IsOpen;

}

set

{

if (value)

{

port.Parity = System.IO.Ports.Parity.None;

port.DataBits = 8;

port.StopBits = System.IO.Ports.StopBits.One;

port.Open();

port.WriteLine("\n\r\x04\r");

System.Threading.Thread.Sleep(1000);

port.WriteLine("from microbit import \*\r");

port.WriteLine("import radio\r");

port.WriteLine("radio.on()\r");

port.DataReceived += Port\_DataReceived;

} else

{

port.Close();

}

}

}

public event ReceivedDataHandler ReceivedData;

public class ReceiveLineEventArgs : EventArgs

{

public string DataReceived;

}

public delegate void ReceivedDataHandler(object sender, ReceiveLineEventArgs e);

protected string lineReceived = "";

private void Port\_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)

{

lineReceived += port.ReadExisting();

if(lineReceived.Contains("\r"))

{

string []parts = lineReceived.Split('\r');

if (ReceivedData != null)

{

for(int i = 0; i < parts.Length - 1; i++)

{

ReceiveLineEventArgs args = new ReceiveLineEventArgs();

args.DataReceived = parts[i];

Match m = Regex.Match(args.DataReceived, "'(-?\\d+)'");

//if(m.Success)

//{

// currentDistance= int.Parse(m.Groups[1].Value);

//}

//ReceivedData(this, args);

}

lineReceived = parts[parts.Length-1];

}

}

}

/// <summary>

/// Switches the front lights on or off

/// </summary>

/// <param name="lights"></param>

public void SetHeadLights(bool lights)

{

send("H" + (lights ? "1" : "0"));

}

/// <summary>

/// Send radio control via the micro:bit

/// </summary>

/// <param name="radiostring"></param>

protected void send(string radiostring)

{

port.WriteLine("radio.send(\"" + radiostring + "\")\n\r");

}

/// <summary>

/// Switches the rear lights on or off

/// </summary>

/// <param name="lights"></param>

public void SetBrakeLights(bool lights)

{

send("B" + (lights?"1":"0"));

}

//ultrasound sensor currently out of use

//protected int currentDistance;

//public int GetDistance()

//{

// if(Connected)

// {

// send("Gd");

// port.BaseStream.Flush();

// port.WriteLine("radio.receive()\n\r");

// port.WriteLine("radio.receive()\n\r");

// port.BaseStream.Flush();

// return currentDistance;

// }

// return 0;

//}

/// <summary>

/// Switches the horn on or off

/// </summary>

/// <param name="horn"></param>

public void SetHorn(int frequency)

{

send("\*" + frequency);

}

/// <summary>

/// Send custom command. Should not be used in production - just for test purposes.

/// </summary>

/// <param name="command">Command to send</param>

public void Send(string command)

{

send(command);

}

/// <summary>

/// Switches the door open motor on or off

/// </summary>

/// <param name="doors"></param>

public void OpenDoors(bool doors)

{

send("D" + (doors?"1":"0"));

}

/// <summary>

/// Sets the speed (from -1023 to +1023). 0 will stop the car

/// </summary>

/// <param name="speed">Speed of the car (-1023 to 1023)</param>

public void SetSpeed(int speed)

{

send("V" + speed);

}

/// <summary>

/// Sets speed to 0 and direction to Forwards

/// </summary>

public void Stop()

{

SetSpeed(0);

Steer(Direction.Forwards);

}

/// <summary>

/// Directions the car can steer

/// </summary>

public enum Direction

{

Forwards, Left, Right

}

/// <summary>

/// Sets the car direction (assuming it's going forwards)

/// </summary>

/// <param name="direction">Direction.Forwards, Direction.Left or Direction.Right</param>

public void Steer(Direction direction)

{

string d = "F";

switch(direction)

{

case Direction.Forwards:

d = "F";

break;

case Direction.Left:

d = "L";

break;

case Direction.Right:

d = "R";

break;

}

send("S" + d);

}

}

}