## Development Environment and Tools

For the majority of the development of this project, the development environment being used will evolve with the requirements of the project. That is to say, in the initial stage it is very likely that a combination of on-board development tools (such as the Vim and Emacs text editors, in addition to Gedit if a graphical session is available) and on-pc development tools (such as Visual Studio Code, which is favourable for this project due to its tight GitHub integration). The use of development environments that are tightly integrated with Git is a priority for this project, as GitHub will be used both for planning the development itself (with code versioning being used), in addition to GitHub serving as the project planning solution through the utilisation of Github Tools.

### Management Tools

For the management of software development in this project and due to the iterative nature of the design process of our particular set of deliverables, an Agile method (Scrum) has been chosen. This fits well with industry standard development methodologies and has a number of effective pre-developed solutions that allow targets to be effectively managed and met. The scrum methodology used will be modified to heavily utilise ‘feature-based’ sprints in contrast to the traditional ‘time-based’ sprints of the Agile methodology, in order to adapt to a development team that will be working with less time resources than a traditional set of company employees. In order to effectively manage the sprints we will be conducting it is integral to consider a number of management solutions. In our case, this will include a package or ‘suite’ of development solutions, *Github Projects*. A list of tools that will be utilised within Github Projects is listed below;

#### Kanban Charts

*Github Projects* allows the use of Kanban Charts, attached to specific targets in GitHub. The main advantage of using Kanban Charts integrated with Github Projects is the integration offered between the charts and the Version Control System (VCS). In integrating Kanban Charts with the VCS, the Issue Tracker and Project Milestones lists can be synchronized with specific targets in Kanban, which makes the use of the charts more flexible than a competing, independent tool like *Trello.* By assigning specific targets to specific timeframes in the project, the Kanban Charts can also be used to determine if the project is on track to be completed in the given timeframe, independent on the success of feature-based sprints.

#### Issue Tracker

The Issue Tracker (also known as the *Issue Council*) is a vital tool not only exclusive to *Github Projects* but also to *Github* as a site, and is often used in Open-Source projects to rectify bugs and manage feature requests. The issue tracker allows a user to create a particular problem or epic, which can be resolved through a commit (a particular feature implementation in the Version Control System), or through closing the issue as having being fixed either in an upstream branch or in a recent commit, or closing the issue as not relevant to the project. The issue tracker will allow a separate channel of communication for the concerns relating to issues in the software, such that the *Kanban Chart* can be focussed entirely on feature implementation.

#### Repository Wiki

Of particualar use for documentation purposes in the project will be the maintenance of a documentation *Wiki*, which in our case will be maintained through *Github Projects*. While serving as a potential guide of deployment for the client and as a sort of ‘Almanac’ to common problems and solutions found along the road of development, the Wiki will also serve as a tool for the developers themselves, such that they are subscribing to the simple principle of DNRY (Do Not Repeat Yourself). By documenting all issues and trials along the development process, we can prevent the same issue from tripping the development team up repeatedly.

#### Version Control System

Probably the most important tool for the development of this project will be the Version Control System (VCS), in our case *Git* will be used, however all competitors provide good solutions such as *Subversion* and *Mercurial*. A version control system allows all source files in the tree to be *Revision Controlled*, which effectively allows iterative development by facilitating the changing of source files with tags, and with historical data for each developer. One important thing this allows us to do is revert features that either do not work, or introduce a particular bug into the system. The VCS is an indispensible tool for a large amount of development, particularly in our case.

### Development Tools

Due to the embedded nature of this project, in addition to the management tools used there will be a distinctly broad selection of development tools required to realise the project. Each of the development tools fit under distinct categories, and will be relevant at different stages in the project. It is important to note at this point that a number of the development tools mentioned in this document may be subject to change as the project and requirements evolve, and some tools may not eventuate to be as useful as originally anticipated.

#### Software (IDE and Operating System)

When the project was forseen to utlise more embedded solutions, it was likely that microcontroller compilers would be needed, in addition to an IDE similar to *Arduino Studio*. As the requirements have changed however, it has become more apparent that it is likely we will always be working in a desktop environment, whether that be on the embedded devices such as the *Beaglebone Black* or *Raspberry Pi* or on a cross-compiled solution such as using *QEMU* on a desktop machine. In either case, the table below indicates the particular environment tools that are anticipated and their intended purpose;

|  |  |  |
| --- | --- | --- |
| **Environment Tool** | **Nature of Tool** | **Intended Purpose** |
| Visual Studio Code | Integrated Development Environment (IDE) | * Development of Application Code (C++ and Python) * Development of Web Management Interface (Anticipated in Python) |
| Linux (*Debian)* | Operating System | * Allows cross-compilation of ARM-compatible code via *QEMU* * Tight integration with all development tools on embedded devices * Used on embedded devices *Raspberry Pi* and *Beaglebone Black* |
| QEMU | Architecture Emulation Layer | * Cross-compiling large programs (Example: OpenCV Libraries for *BeagleBone* and *Raspberry Pi*) * Utilising computing power of desktop systems to accelerate embedded compilation |
| Vim & Bash | Scripting & Text Editors | * Development on-device * All build systems (build systems will be customized via *bash*) |

#### Hardware Solutions

It would be impossible to realise this project without an amount of hardware solutions, which complement the software that has been writted for them. In our case, this is still a point of revision and prototyping, with a quite strict set of devices prototyped but no eventual solution decided upon. At this stage, it is likely that the multicore advantages of the *Raspberry Pi* will be sufficient, however more prototyping is necessary in order to make a final decision on the particular hardware stack that will be employed. In addition to the deliverable hardware, tools related to electronics prototyping are also included in the below matrix;

|  |  |
| --- | --- |
| **Hardware** | **Potential Use** |
| Raspberry Pi | * Processing of camera images into an object detection algorithm (possibly *YOLO* or *TensorFlow*) * Sending of image data over the network * Raw sensor reading (custom sensors) through the *Analog Inputs*, each of which supporting a 16-bit *ADC* |
| BeagleBone Black | * Processing of camera images into an object detection algorithm (Probably an older, more optimised solution – HAAR Cascades or HOG Classification with Non-Maximum Suppression) * Sending data over the network and hosting web pages * Reading sensors through an even larger set of ADC than the Raspberry Pi can provide |
| Google Coral USB Accelerator | * Accelerating Raspberry Pi processing of images using the *TensorFlow* machine learning library |
| USB Webcam | * Input of images for either the *BeagleBone* or *Raspberry Pi* * High latency, however higher quality than the alternative, the *Pi Camera*. |
| Pi Camera | * Input of images for either the *BeagleBone* or *Raspberry Pi* * Lower latency, in exchange for a **lower camera quality**. * Lower camera quality could accelerate image processing through the chosen algorithm |
| Breadboarding Tools | A selection of breadboarding tools such as a breadboard, connectors and basic components (such as resistors or capacitors) can provide options for prototyping board couplings. |
| Various Sensors | In order to comply with stretch requirements, sensor data could be collected through the embedded device **analog port** and collated with traffic and object detection data. |