# 1. Project Introduction

My team decided to create a remote-controlled hazard detector using a range of technologies and sensors. The following technology and sensors were chosen:

1. MKR Wi-Fi 1010 Arduino board wired to sensors that streams the data to a WebSocket server hosted in Python.
2. Zumo 32u4 with motor libraries and Xbee remote that takes in data over the serial port from a Python GUI.
3. Light Sensor wired to the MKR Wi-Fi 1010 analogue pin.
4. Temperature Sensor wired to the MKR Wi-Fi 1010 digital pin.
5. Geiger counter wired to the MKR Wi-Fi 1010 digital pin.
6. Smart phone camera being streamed to a locally hosted server ([IP Camera App](https://play.google.com/store/apps/details?id=com.pas.webcam&hl=en_GB&gl=US))
7. A python-based graphical user interface using [Tkinter](https://docs.python.org/3/library/tkinter.html) (general UI), [OpenCV](https://pypi.org/project/opencv-python/) (displaying the live webcam feed from the locally hosted server), [PySerial](https://pyserial.readthedocs.io/en/latest/) (Zumo XBEE control) and Threading to run all three concurrently.

When brought together this created a robot that could be remotely controlled via a Python Tkinter interface that can detect light, temperature and radiation within a room and report back whether it is safe to be inside.

My role in this project focused on the Python interface, hosting the Web-socket server, using the Geiger counter to get counts per minute data and using a smart phones camera to get a feed. These will be further elaborated in this report.

# 2. Project Aims

The aims for the project are:

* Remote control of the Zumo via Xbee using Python Interface
* Remote control of the sensors (Geiger, Light and Temperature) via an MKR 1010 Arduino WIFI board using a WebSocket server to stream continuous data to a Python GUI.
* Remote streaming of a camera module that will give a wireless display of the robot to navigate clearly from a distance.
* A Python graphical interface with threads that allow for control of the Xbee, display of a video camera feed and display of streamed data from a WebSocket on the network.
* A housing for all the sensors and a way to move them around using a Zumo.

# 3. Team Project Management and progress tracking

The team kept track of the project progress by using a Kanban agile approach. Using [Trello](https://trello.com) allowed us to write “notes” for each task and assign them to different project members. In Appendix A1 the Trello board is shown.

The team took a scrum approach with assigning a certain number of notes for the week. We would have a meeting each day to discuss our progress and whether we would require more time. This meant that the team was always kept up to date with the progress of the project.

# 4. Project Development

## 4.1 Geiger Counter and sensors

One of the main sensors for this project is the Geiger Counter. This works by using a sealed tube with gas that when radiation collides with the gas, pushes an electron away from the gas atom and creates an ion pair. A wire in the middle of the tube attracts the electrons, creating other ion pairs and sending a current through the wire (nrc.gov, 2020).

Getting the Geiger counter to work was simple; after connecting it to digital pin 4 on the MKR 1010 and each time radiation collides with the gas it would turn pin 4 high. We used a program to count how many collusions occurred in a minute (CPM). This count is how radiation is measured and gives a reading on how safe the area is.

The second sensor in use was a simple LDR (light dependant resistor) which changes its voltage based on the light levels in the room. The output of this was connected to the analogue pin on the MKR 1010.

The third sensor in use was a digital temperature sensor. The original plan was to use an analogue version, however the MKR 1010 was creating too much interference and causing a large variance in the temperature readings. The output of the digital temperature sensor was connected to a digital pin on the MKR.

The sensor information is sent to a WebSocket server that is then sent to the GUI to be displayed in real time. This WebSocket server is hosted on the network, which means that the laptop with the GUI and the MKR must be connected to the same network to work.

## 4.2 IP Camera

The original plan had intended to utilise the [OV7670](https://circuitdigest.com/microcontroller-projects/how-to-use-ov7670-camera-module-with-arduino) Arduino camera module, however the order did not arrive in time for the project. To get around this it was decided that a mobile phone could be set up to become an IP Camera. This is where a phone hosts a server that another device can connect with to view the footage.

Using the mobile camera added extra internet of things functionality and meant that we could move the robot into another room and still be able to navigate it. While working with this server I could connect to it via the python module OpenCV.

This worked by getting each frame of the video feed and continuously updating it onto a Tkinter canvas variable each millisecond. This made the camera feed show up in real time on the GUI and could be used for navigation.

The skeleton of the video code was used from [this](https://solarianprogrammer.com/2018/04/21/python-opencv-show-video-tkinter-window/) tutorial. Extra adaptations were made to add an overlay of the sensor values, as well as adding extra classes for the WebSocket sensor feed and a class to add keyboard inputs to send via the Xbee to the Zumo.

## 4.3 Robot Chassis

In appendix A2 an image displays the final chassis for the robot. This was constructed using Lego that houses the electrical components of the phone, MKR 1010, breadboard, Geiger Counter (on the underside), mobile phone and the battery bank that powered everything. I did not have a big role in the construction however did suggest using elastic bands to hold certain items in place. In a real device this would be secured in place using nuts and bolts and have a better distribution of weight. Even though my role was minimal I was kept up to date with the progress of the build.

## 4.4 Python Graphical User Interface

The largest part of the project was creating the Python GUI. I was new to using threading, as well as the graphical modules OpenCV and Tkinter so had to learn how to use these effectively. As mentioned above a skeleton code was used for the displaying of the video, however I added extra classes, functions, and variables to accommodate the extra functionality of the Zumo control and the livestream of sensor values.

The python GUI was split into three main sections:

1. A thread that controlled the Zumo using the PySerial module. This connected to the Xbee module on serial port 9600. This part of the code takes in keyboard inputs ‘w’ (forward), ‘a’ (left), ‘s’, (backwards), ‘d’ (right) which are sent wirelessly to the Zumo to control its movement.
2. A second thread that hosted a WebSocket server on port 5000 of localhost. They received sensor values every second which the Python GUI took in and updated a dictionary that contained the values.
3. A helper function that constantly compared the new temperature and counts per minute values then set the new max values as we wanted to keep a track of the peak values during a mission.
4. A helper function that would change the colour of the labels as well as give a danger level to inform users to whether the robot was in a safe or dangerous area and alerting to what type of hazards are present.
5. A class that displays the video feed with the labels added as an overlay. This is the skeleton code that has been adapted from [this](https://solarianprogrammer.com/2018/04/21/python-opencv-show-video-tkinter-window/) tutorial. This also gave the functionality of saving screenshots of the video that is saved onto the computer that can be viewed after the mission.

# Contributions to the team and project

The project development section goes into details of each aspect and my contribution to each area. I lead the team by creating the Trello board and the tickets as well as delegating tasks to team members.

The team worked autonomously for the most part; after we had our tasks we worked on them until we finished and then would pick up the next available tasks. It was my idea to use a Geiger Counter so I purchased one and I learnt how to use the Geiger counter using an [instructibles](https://www.instructables.com/Arduino-DIY-Geiger-Counter/) tutorial and the code was adapted to the Arduino sensor code.

The primary focus of my work was the Python GUI and learning how to use threading. Threading allowed for multiple tasks to be carried out at once in the program, and this took a while to learn.

I also worked on the IP Camera functionality after we were unable to get access to the Arduino component in time. This was simple to setup by installing an app however it was difficult to connect to integrate into the Python code. This is because the IP’s changed with every new network, and the asynchronous python module [asyncio](https://docs.python.org/3/library/asyncio.html) is not compatible with the [threading](https://docs.python.org/3/library/threading.html) module.

I found a work around where I encased the entire WebSocket code in a thread so that it was self-contained. This allowed for a continuous flow of data from the sensors sent from the MKR 1010 WI-FI board.

# Conclusion

In conclusion the project was largely a success. If there was extra time, a more robust chassis would have been created for the robot. The Zumo struggled to turn and grip on smooth surfaces and performed better on carpet because there was more friction.

A second improvement would be to use the originally intended [OV7670](https://circuitdigest.com/microcontroller-projects/how-to-use-ov7670-camera-module-with-arduino) camera. This is much lighter and compact which would reduce weight and would mean that the IP Camera server would not be required. All the data could have been sent across from the MKR 1010 Wi-Fi module.

My team worked well using an agile methodology, if we were to do this project again, I would still use the Kanban structure. In future I would have allocated more time for the work tickets to be done duo to a couple issues with sickness within the group which delayed us.

If we had considered likely disruptions, we could have reduced the rush at the end of the project where we nearly didn’t complete in time. It would have given room for more features such as a more user intuitive interface with graphical images to display the light, radiation, and temperatures.

I feel that I had a big role within the team, helping organise and delegate tasks as well as facilitating effective communication in the team. Given another opportunity I would have encouraged my team members to try a more proactive approach rather than rely on being told which tasks to take.

### Appendix A

### Graphical user interface, application Description automatically generated

Appendix A1 Trello BoardA picture containing electronics, circuit

Description automatically generated

Appendix A2 Geiger Counter

### 

### 

Appendix A5 Closer top view robot chassis

Appendix A4

# Bibliography

nrc.gov. (2020, march 19). *What is a Geiger Counter?* Retrieved from U.S.NRC: https://www.nrc.gov/reading-rm/basic-ref/students/science-101/what-is-a-geiger-counter.html

Top view robot chassis