Electronics and Computer Science

Faculty of Engineering and Physical Sciences

University of Southampton

Yaqin Hasan

2nd May 2023

An Intelligent Approach to Navigating Environments With Compliant Obstacles

Project supervisor: Dr. Danesh Tarapore

Second examiner: Dr. Iain McNally

A project report submitted for the award of

MEng Computer Science

# Abstract

Obstacle compliance is a property that determines how easily an obstacle can be moved or otherwise navigated through. When navigating environments with compliant obstacles, such as outdoor forest environments, it is important to know how compliant each obstacle class is.

This is a challenge for vision-based autonomous navigation systems due to it being difficult to ascertain compliance based on visual data. Physical compliance testing is therefore a necessity in a lot of cases. However, it would be ideal that after the testing is complete, the system learns and uses this data for future navigation decision making.

This project aims to develop working prototype hardware and software to demonstrate that obstacle compliance can be reliably and autonomously determined using accelerometer data from an IMU. Additionally, it must also be able to use data from previously performed tests to make future navigation decisions.

The final prototype was able to meet the aforementioned goals by autonomously testing the compliance of novel obstacles with unknown compliances as well as inform the user if it encounters an obstacle with a known compliance value.

# Statement of Originality

**Statement of Originality**

* I have read and understood the [ECS Academic Integrity](http://ecs.gg/ai) information and the University’s [Academic Integrity Guidance for Students](https://www.southampton.ac.uk/quality/assessment/academic_integrity.page).
* I am aware that failure to act in accordance with the [Regulations Governing Academic Integrity](http://www.calendar.soton.ac.uk/sectionIV/academic-integrity-regs.html) may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
* I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

***You must change the statements in the boxes if you do not agree with them.***

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

| **I have acknowledged all sources, and identified any content taken from elsewhere.** |
| --- |

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

| **I have not used any resources produced by anyone else.** |
| --- |

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

| **I did all the work myself, or with my allocated group, and have not helped anyone else.** |
| --- |

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

| **The material in the report is genuine, and I have included all my data/code/designs.** |
| --- |

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

| **I have not submitted any part of this work for another assessment.** |
| --- |

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements*.* You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

| **My work did not involve human participants, their cells or data, or animals.** |
| --- |

*ECS Statement of Originality Template, updated August 2018, Alex Weddell* [*aiofficer@ecs.soton.ac.uk*](mailto:aiofficer@ecs.soton.ac.uk)

# Contents

[**Abstract 2**](#_gjdgxs)

[**Statement of Originality 3**](#_30j0zll)

[**Contents 5**](#_3znysh7)

[**Introduction 6**](#_3dy6vkm)

[**Background Reading 8**](#_wumbv7v353vq)

[**Analysis and Design 9**](#_2s8eyo1)

[Simulator and Hybrid Approach: 9](#_665aiq26qap6)

[Real-World Approach 10](#_n690v7xqqu0u)

[**Implementation 12**](#_796od9zb5ffx)

[Stage 1: Pi Camera and Thymio Interface Testing 12](#_5jm8s21p23pf)

[Pi Camera Setup: 12](#_bh5xirmki7ga)

[Thymio Interface Setup: 13](#_gw52h0r9sxsj)

[Stage 2: Exploring Inertial Measurement Units 14](#_uzfwcuvi5hz3)

[Figure 1: Insufficient sample rate resulting in missed spike in signal 15](#_7ykwh4dso7w3)

[Stage 3: The MPU6050 16](#_4mblmre3zlto)

[Figure 2: Raw IMU data at rest 17](#_kpt0k2xwzjyw)

[Stage 4: OpenCV 18](#_8s4zmvg95pmz)

[OpenCV Setup: 19](#_kswo1vitldwm)

[Object detection and classification code: 20](#_m849asvwobag)

[Stage 5: Live Testing 21](#_tcm4md922zqx)

[Figure 3: Raw IMU data after preprogrammed test 21](#_egc1raclf33k)

[Stage 6: Prototyping 22](#_otj0s6fv5vy3)

[Physical design 22](#_lp6tmphtv3qt)

[Final Code Implementation Breakdown 22](#_dpq54kizc3qe)

[Detailed Breakdown of Compliance Testing 24](#_q918dlwliqgf)

[Figure 4: A simplified illustration of a frame based approach 24](#_vi8aic2b1dhg)

[Figure 5: A simplified illustration of the new approach 25](#_hg7g7okeep5x)

[**Testing Methodology 26**](#_h49wmrx169k)

[**Conclusion 27**](#_3rdcrjn)

[Figure 6: PERT chart from progress report 27](#_pa4mkre3otbo)

[Potential Future Developments 28](#_t0d3428rmtuc)

[**References: 29**](#_yal24b49901z)

[**Appendices 31**](#_yb9fcnop2c5s)

[Appendix A: GitHub Repository for ThymPi Project 31](#_yjpuc43ut2t2)

[Appendix B: Hardware Costs 31](#_35nkun2)

[Appendix C: Photograph of Testing Setup 32](#_jl7nsz9w38j9)

[Appendix D: Initial Project Brief 33](#_tebu1rmjaa30)

# Introduction

I would like to begin this report by stating that the project source code is available on GitHub at ***https://github.com/yaq1n0/ThymPi*** as minor changes may be made to it over time. The link to the repository is a substitute for having the code in the appendix of this report. Additionally, files from this repository are referenced as follows ***ThymPi/example\_dir/example\_file.py*** at numerous points throughout this report. Lastly, this repository also contains a ***README.md*** file which contains information on how to get started with the code.

Autonomous navigation is an enormous and rapidly developing field. With outdoor environments being a particular challenge for autonomous robots to navigate for multiple reasons, one of which being the presence of compliant obstacles.

Compliant obstacles are obstacles that can be moved or navigated through. In this context, we can define compliance on a spectrum from completely compliant to partially compliant to completely non-compliant.

Sensor-based navigation is one of the approaches used in autonomous navigation. Sensors can be loosely broken down into two categories, internal and external. Internal sensors sense the state of the robot (accelerometers, gyroscopes, PWM, etc). External sensors sense the state of the environment (cameras, proximity, microphone, etc).

Navigating with purely visual information is of great interest due to the rapid development of machine vision technologies. These allow for great flexibility and ease in developing autonomous navigation systems.

However, compliant obstacles pose a challenge to purely visual autonomous navigation systems as using only visual data it can be difficult to determine how compliant an obstacle is.

One possible solution would be to pre-program the compliance of known obstacles, however this consumes a lot of human hours and has no recourse in the case of unexpected obstacles.

A better solution would be to use other sensors to determine compliance at runtime and supplement the visual data. Ideally, this could be done such that the system tests an obstacle once and learns its compliance in order to make future navigation decisions.

This project aims to develop working prototype hardware and software to demonstrate that obstacle compliance can be reliably tested using onboard sensors and used for future navigation. In order to demonstrate this, the prototype must demonstrate the capability to autonomously perform compliance testing on an obstacle. Additionally, the compliance testing must reliably be able to determine the object’s compliance based on sensor data, with a clear correlation between the determined compliance value and the actual compliance of the obstacle. Lastly, the system must also demonstrate that past testing data can be used for future navigation decisions.

Due to this project being both heavily time and resource constrained, a very precise scope must be defined. Even when focusing on just developing an autonomous navigation system for outdoor forest environments, there is a lot that must be abstracted in order to produce something within the constraints of an undergraduate final year project.

It would not be practical to develop and test in an actual forest environment, therefore an analogous test environment with obstacles of varying compliance would be used.

Furthermore, an appropriate physical platform for prototyping must be chosen. It wouldn’t be practical to use a large and expensive outdoor or forest rover for this project. Therefore, a Thymio robot was used in this project [(*Home*, 2021)](https://paperpile.com/c/0G2eWB/yLt2). This robot was loaned to me by the university and is a good fit for multiple reasons. The Thymio framework and supporting resources being the primary reason, with it allowing for a relatively easy development process.

Initially, it was thought that the final prototype would encompass the robot autonomously navigating through an obstacle course with compliant obstacles. This would demonstrate that the robot was able to test an object's compliance reliably and use that information to supplement the navigation algorithm.

However, in the final prototype, the navigation aspect of the demo was scrapped as it made no difference to the actual demonstration of project goals as described above of compliance testing and using that data in the future.

# Background Reading

Autonomous vehicles alone bring in billions of dollars of investment annually [(Koetsier, 2023)](https://paperpile.com/c/0G2eWB/cH3e). While this makes up a large portion of the autonomous navigation field and certainly garners the most publicity for it, it isn’t the entire field.

Autonomous rovers, underwater-rovers and drones are another large part of the field. In this report, autonomous forest rovers are of particular interest as they are the likely to encounter compliant obstacles.

The University of Southampton has a history of undergraduate projects related to navigation in forest environments. The first project of interest is by Matthey Brako regarding *Vision-Based Navigation for an Autonomous Rover in a Forest Environment* [*(Brako, 2022)*](https://paperpile.com/c/0G2eWB/G8Qm). This is a good example of what a vision based navigation system might look like, as mentioned in the *Introduction* section.

Another of interest is by Callum Newlands regarding *Generating and Rendering Realistic, Navigable Forest Environments* [*(Newlands, 2021)*](https://paperpile.com/c/0G2eWB/EtNl). In the very early stages of this project, it was considered that this simulator could’ve been extended to account for obstacle compliance and used to implement this project.

Another simulation suite that was considered during the very early stages of this project was argos3 [(*The ARGoS Website*, no date)](https://paperpile.com/c/0G2eWB/oq7y). argos3 is a physics-based robot simulator that is commonly used in swarm robotics to simulate large swarms efficiently. The physics-based nature of argos3 was of particular interest as it would make for easier implementation of compliance.

There’s also a version of argos3 developed for the Thymio which would’ve allowed for development within a simulated environment to directly be ported over to the real world [(*thymio: Thymio simulator based on ARGoS framework*, no date)](https://paperpile.com/c/0G2eWB/DT02).

These aforementioned projects are of particular interest as they could all be integrated with this project to some degree. For example, combining a low-viewpoint visual navigation algorithm with reliable runtime compliance testing would allow for autonomous navigation of an environment with compliant obstacles such as the forest floor.

## 

# Analysis and Design

This section will detail the analysis stage and initial design of the project, however it should be noted that changes to the final design were made throughout the implementation stages of the project. There were also multiple approaches that were being considered simultaneously during this stage as will be detailed below.

### Simulator and Hybrid Approach:

As previously mentioned in the *Background Reading* section, an approach involving a simulated environment was considered. With two different simulators considered.

The first simulator considered was the forest simulator by Callum Newlands [(Newlands, 2021)](https://paperpile.com/c/0G2eWB/EtNl). This was the first simulator found during the research phase and was thought of to be a good fit due to being used by Matthew Brako in his project regarding vision based navigation.

However, this was quickly dismissed as this simulator was clearly designed with vision-based navigation approaches in mind. And implementing collision physics required to simulate compliance would be quite difficult.

On the other hand, the physics-based Thymio simulator [(*thymio: Thymio simulator based on ARGoS framework*, no date)](https://paperpile.com/c/0G2eWB/DT02) seemed much more promising as it already had a lot of the physics implemented. Furthermore, this also allowed for deployment of simulator code to the real-world, which would allow for both a simulated and real-world demonstration.

With this approach, the idea was to demonstrate that the simulated robot would be able to test obstacle compliance and demonstrate the project goals within a simulated environment. Then possibly slightly modify and then deploy the code onto a real-world robot for further demonstrations.

Eventually, this approach was dismissed for two main reasons. Firstly, the Thymio simulator code was difficult to set up due to the code being multiple years old and sparsely documented. Even when working correctly, it would be quite inconsistent and sensitive to changes in the software environment that it was tested in. Secondly, the code relied on a lot of obfuscated dependencies, and would’ve made implementation of the required additional functionality quite difficult.

### 

### Real-World Approach

A purely real-world approach was also simultaneously considered, both as a complement to the aforementioned hybrid approach as well as on its own. As mentioned in the *Introduction*section, we would use the Thymio robot as the platform of choice.

The Thymio is a small (~roughly 10x10 cm) robot with 2 wheels and a multitude of sensors and actuators on board. However, it doesn’t have any innate compute capability beyond following simple pre-programmed scripts, which means that a computer has to send control signals to it.

On a side note, there is a version of the Thymio which is capable of receiving wireless control signals, which would be an interesting path to pursue as it would allow for the platform to shed the onboard computer and its power supply. However, the Thymio loaned to me by the university only had wired capability and therefore would have to be controlled by a computer small and light enough to be mounted on it alongside a power source for it.

The Raspberry Pi is a very well known and supported single board computer, which would be quite familiar to the majority of ECS staff and students. I personally was very familiar with the Raspberry Pi due to a mixture of personal projects and university modules involving it. In fact, the specific Raspberry Pi 3B+ used in this project was actually given to me by the University for Computer Systems I during my first year.

There would also need to be a way to power the Raspberry Pi, with the main requirement being that the power supply must be capable of 5 Volts and 2.4 Amps or more (ideally 3 Amps). Most commercially available battery packs are only capable of 2 Amps, with ones capable of high current usually being larger (>= 20,000mAh capacity) and therefore too heavy to be feasibly mounted on a Thymio. Eventually, A 5000mAh power bank with 5 Volt 3 Amp current capability was purchased off Amazon which fit the requirements of this project.

The Thymio has a variety of onboard sensors, including an accelerometer, which will be discussed further in the *Implementation* section. However, a notable exception to this is the lack of a camera. While this project doesn’t necessarily require a camera to strictly meet the project goals, one of the potential extensions or future uses of this project is to supplement a vision-based autonomous navigation system. Therefore, it would be useful to be able to tie in visual data to some degree. The hardware selection for this was actually fairly straightforward as the Raspberry Pi Camera is a very inexpensive and well-documented piece of hardware with many pre-written libraries that make development for it extremely straightforward.

Ideally, a multi-sensor approach would be used to determine compliance. With the following sensors being identified as potentially being useful to determine obstacle compliance via the following proposed methods.

* A high precision GPS could be used to determine position and therefore speed, showing a slowdown or stop depending on object compliance.
* A wheel tachometer could be used in conjunction with measuring the PWM signal or DC voltage of the wheel motors, with lower wheel speed for a certain amount of wheel power being indicative of lower object compliance.
* A pressure sensor could directly be used to determine how hard the robot is pushing on an obstacle and therefore how compliant it is.
* An accelerometer could be used to determine object compliance based on the deceleration caused by the collision/

However, due to the time and resource constraints of this project, it was determined that using a single sensor would be the most feasible despite the reduction in precision and accuracy of the final prototype. We concluded that a single sensor approach would still be sufficient to prove the concept that sensor data can be used to determine compliance reliably. Furthermore, there is also the possibility of using the lessons learnt during this project to extend the project further to incorporate a multi-sensor approach as will be further discussed in the *Conclusions*section of this report.

# Implementation

In this section, all efforts were taken to attempt to follow the chronological order of the implementation process. However, the design and implementation process is non-linear with tasks often being done in parallel with many of them being dependent on each other. Therefore, this section has been broken down into subsections that loosely follow chronological order but give precedence to readability.

## Stage 1: Pi Camera and Thymio Interface Testing

### Pi Camera Setup:

In this project, the Pi Camera v1.3 was used, although different cameras such as the Pi Camera v2 would work with small adjustments. The Pi Camera v1.3 is a 5MP sensor capable of 1920x1080@30fps [(*Raspberry Pi Camera V1.3 —*, no date)](https://paperpile.com/c/0G2eWB/fldY).

*Note: The Pi Camera v1.3 works best with Debian 10 “buster” based linux distributions*

* Enabling the Raspberry Pi camera interface
  + Type ***sudo raspi-config*** into the command line
  + Go to *interface options > camera*
  + Enable the Raspberry Pi camera interface
  + Exit raspi-config and reboot the Raspberry Pi
* Test the camera in command line
  + Type ***raspistill -o image.jpg*** into the command line
  + Check image.jpg
* Setting up the python picamera library
  + Type ***sudo apt-get install python-picamera python3-picamera*** into the command line
  + OR
  + Type ***pip install picamera “picamera[array]”***

After setting up the Pi Camera, ***tests/unit\_tests/cam\_test.py*** was created in order to test the camera using a python interface.

### 

### Thymio Interface Setup:

In order to interface with the Thymio using the Raspberry Pi 3B+, the following setup steps have to be completed [(Hofer, no date)](https://paperpile.com/c/0G2eWB/8Fzf).

* Downloading and Installing ASEBA and its dependencies
  + Download ***aseba\_1.5.5\_armhf.deb*** from ***http://wiki.thymio.org/en:linuxinstall***or a later compatible version if available.
  + Move the file to the working directory
  + Enter the following commands into the command line
    - ***sudo dpkg -i aseba\_1.5.5\_armhf.deb***
    - ***sudo apt-get update && sudo apt-get -f install***
    - ***sudo apt-get install python-dbus***
    - ***sudo apt-get install python-gtk2***
* Start the Asebamedulla service using ***asebamedulla ser:name=Thymio-II***
* Stop the Asebamedulla service using ***pkill -n asebamedulla***

Following lebalz’ guide on GitHub [(Hofer, no date)](https://paperpile.com/c/0G2eWB/8Fzf), ***ThymPi/tests/unit\_tests/thymio\_test.py*** was created in order to test sending control signals to and reading sensor data from the Thymio.

## 

## Stage 2: Exploring Inertial Measurement Units

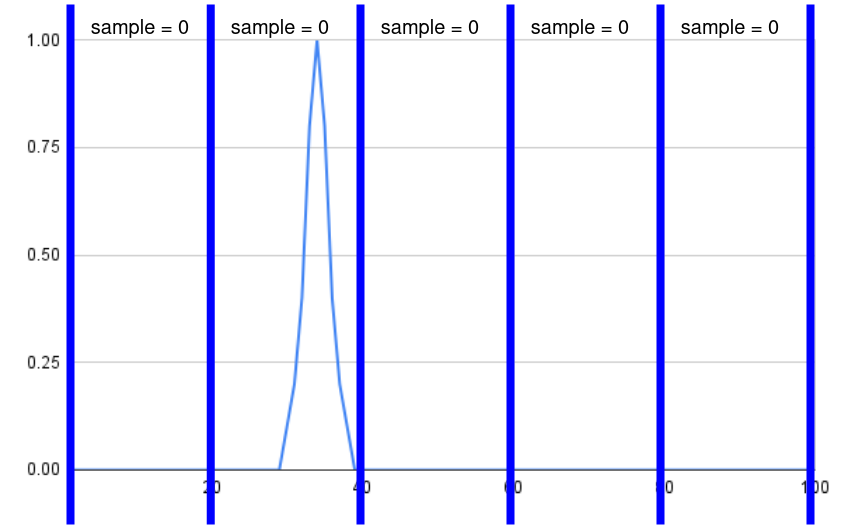
An Inertial Measurement Unit (IMU) is a sensor class containing both an accelerometer and a gyroscope, that when used together can be used to determine linear and rotational acceleration. This was chosen as the sensor of choice for this project partially due to the Thymio having one onboard.

Additionally, I was in possession of a variation of the Raspberry Pi Pico RP2040 called the Arducam Pico4ML. Among other additions, it has an ICM-20948 Inertial Measurement Unit onboard [(*Arducam Pico4ML TinyML Dev kit: RP2040 board w/ QVGA camera, LCD screen, onboard audio, reset button & more*, 2021)](https://paperpile.com/c/0G2eWB/9B0k). It was proposed that if I was able to read the accelerometer data from the IMU and transmit it over the serial bus to the Raspberry Pi, I would be able to use it as a sensor for this project.

Simultaneously, I was also further exploring the Thymio’s onboard accelerometer which would be an even more straightforward solution. However, I was informed by my supervisor that the data from the Thymio’s accelerometer might be unsuitable for this project as past students have attempted to use it before and encountered issues with precision.

Initially, the Pico4ML showed promise as the accelerometer readings were both frequent and precise. However, the communication over serial ended up being a much bigger hurdle than anticipated. While I was eventually able to get the accelerometer data to show up on the Raspberry Pi, the serial bus wasn’t very consistent and wouldn’t work for a variety of seemingly random reasons during testing.

On the other hand, reading the Thymio’s accelerometer data was much more straightforward and consistent. Unfortunately, the data itself became the source of inconsistency. For context, an important test case is trying to get accelerometer data for collision with a wall or some other completely immovable non-compliant obstacle. The expected data would be a very sharp spike of deceleration. However, I would often not get this expected data during testing, which I then determined to be an issue with the Thymio’s onboard accelerometer having too low of a sample rate.



##### Figure 1: Insufficient sample rate resulting in missed spike in signal

*Figure 1* is a simplified visual representation of the test data that demonstrates how too low a sample rate can result in spikes in a signal being missed. This ended up resulting in the Thymio’s accelerometer being unsuitable for this project.

In hindsight, looking at the sensor readings that Sina Sarparast got during his testing would’ve allowed me to determine that without having to test this on my own [(Sina, no date)](https://paperpile.com/c/0G2eWB/k5FW). Although I kept pursuing this route because I assumed that there would be a way to interpret the data differently to determine compliance.

Furthermore, during the later stages of the project I had discovered the Thymio API documentation [(*Programming interface - thymio & aseba*, no date)](https://paperpile.com/c/0G2eWB/BV7J). Which also would’ve shown that the Thymio’s accelerometer only had a 16 Hz sample rate which is far from being sufficient to reliably detect deceleration events due to a collision. Additionally, the integer -32 to +32 range would’ve also been insufficient precision to calculate compliance for obstacles with similar actual physical compliances.

## 

## Stage 3: The MPU6050

We eventually decided on purchasing a discrete IMU. There were a few requirements based on the lessons learned in the prior stage.

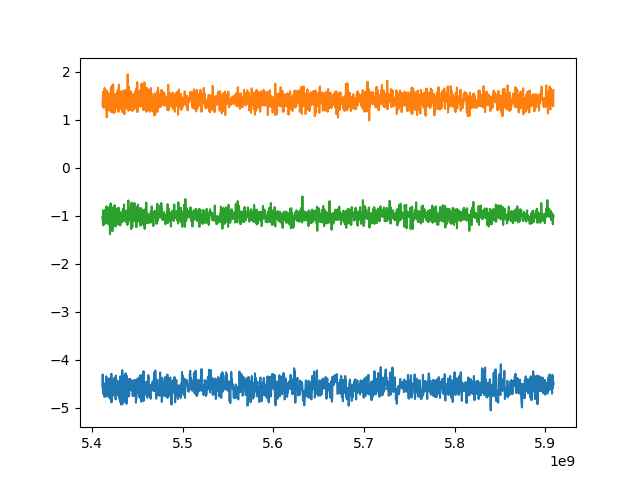
* Simple and reliable interfacing: The IMU data would have to be easily and consistently read by the Raspberry Pi 3B+ unlike the serial interface required by the ICM-20948 on the Pico4ML.
* High precision: The IMU data would have to be precise enough to measure small differences in deceleration amounts in order to account for obstacles with similar compliances.
* High sample rate: The IMU data would have to be frequent enough to be able to detect short spikes in acceleration of deceleration such as those caused by a collision.

After researching several options, the MPU6050 was chosen because it fits all the above requirements and is relatively inexpensive. It communicates over the Inter-Integrated Circuit (I2C) bus, which is very well supported by the Raspberry Pi 3B+’s GPIO pins. It has high enough precision, far above the integer -32 to +32 range of the Thymio’s accelerometer. Lastly, it also has a high enough accelerometer sample rate of 1kHz, which would sufficiently capture sharp spikes in acceleration or deceleration.

While I initially thought of implementing the I2C interface in python myself, I decided to look around to see if anyone had already done this before. As a result, I discovered a simple python module that abstracts away all the I2C setup along with some other things in order to simplify reading the MPU6050’s accelerometer data to a single method [(Martijn, no date)](https://paperpile.com/c/0G2eWB/uBnR).

The installation for this module is quite straightforward and well documented on the GitHub page. However, I would like to note that for this project, I used the second installation method where I ran the ***setup.py*** file to build the module from source as I encountered some issues with installing using pip although this could be an isolated issue.

Once everything was set up, I created ***ThymPi/tests/unit\_tests/imu\_test.py*** which reads the IMU data for a certain testing duration then displays it at the end of the test. The figure below demonstrates what this raw IMU data looks like.



##### Figure 2: Raw IMU data at rest

As you can see in the figure above, the 3 axes measured by the accelerometer are plotted as different colours. This shows the rough amount of noise in the signal as well as what the ranges for the values are when at rest. Knowing this data is important for calibration and tweaking purposes in the future.

## 

## Stage 4: OpenCV

As part of the initial testing methodology, it was decided that it would be a good idea to use the camera to detect and classify obstacles and initiate the automated compliance testing. The first proposal was to use different coloured cardboard boxes with different physical weights in order to simulate objects of different compliance. In this scenario, instead of using any machine vision or object detection and classification software, we would simply detect colour and use that instead of detecting the obstacle’s class.

For example, an immovable red box of weight 2kg would be assigned a compliance value of 0 after testing whereas an easily moveable green box of weight 0.2kg would be assigned a high compliance value of 0.8.

The proposed method of retrieving the box’s colour was also quite rudimentary with the assumption that the coloured box would take up a large portion of the camera’s field of view and frame. A simple frequency analysis would then show the box’s colour disproportionately overrepresented, which would then be used as the key in the key-value pair of colour and compliance.

However, whilst in the process of researching how to implement this, I discovered the following video/article detailing how to use OpenCV to perform object detection and classification on a Raspberry Pi using a Pi Camera [(Tim, 2021)](https://paperpile.com/c/0G2eWB/7wWy).

I followed that thread until I was able to reliably detect and classify objects with my testing setup. After discussing with my supervisor, we concluded that this would still demonstrate the required project goals and therefore this approach was chosen over the initial approach.

This meant that the testing methodology had to change to accommodate this. Using different objects with different physical weights in order to simulate objects of different compliance instead. As a result of the change in approach, we wouldn’t necessarily be completely abstracting away the object detection and classification, but instead be abstracting a significant portion of it instead by using a pre-existing library and a pre-trained model.

### 

### OpenCV Setup:

Following the article by Tim is the best way to get opencv setup [(Tim, 2021)](https://paperpile.com/c/0G2eWB/7wWy). However, it is to be noted that the install process is often very different depending on the specifics of the platform you’re trying to get it installed on and it’s best to figure out how to do so for your specific platform on your own.

Some things that can affect your install process include:

* The computer you’re trying to install OpenCV on
* What functionality of OpenCV you actually need
* The pre-installed dependencies you may of may not have on your operating system

The following is a rough guide of the commands required to install OpenCV:   
***sudo apt-get update && sudo apt-get upgrade***

***sudo apt-get install build-essential cmake pkg-config***

***sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev***

***sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev***

***sudo apt-get install libxvidcore-dev libx264-dev***

***sudo apt-get install libgtk2.0-dev libgtk-3-dev***

***sudo apt-get install libatlas-base-dev gfortran***

***sudo pip3 install numpy***

***wget -O opencv.zip https://github.com/opencv/opencv/archive/4.4.0.zip***

***wget -O opencv\_contrib.zip https://github.com/opencv/opencv\_contrib/archive/4.4.0.zip***

***unzip opencv.zip***

***unzip opencv\_contrib.zip***

***cd ~/opencv-4.4.0/***

***mkdir build***

***cd build***

***cmake -D CMAKE\_BUILD\_TYPE=RELEASE -D CMAKE\_INSTALL\_PREFIX=/usr/local***

***-D INSTALL\_PYTHON\_EXAMPLES=ON***

***-D OPENCV\_EXTRA\_MODULES\_PATH=~/opencv\_contrib-4.4.0/modules -D BUILD\_EXAMPLES=ON ..***

***make -j $(nproc)***

During the installation process, you may also find it difficult to determine if OpenCV has been installed and set up properly as I had. A quick and easy way to check if OpenCV has been installed and set up properly is trying to import the cv2 module and access something within it using python.

### 

### Object detection and classification code:

In the article, there’s a download link to ***Attachment - Object\_Detection\_Files.zip*** [(Tim, 2021)](https://paperpile.com/c/0G2eWB/7wWy). This contains multiple files, of which the following are of interest to this project:

* coco.names
* frozen\_inference\_graph.pb
* ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt
* object-ident.py

The first 3 files are related to the pre-trained COCO (Common Objects in Context) model files. Within the scope of this project, it isn’t important to know much about what these files are or what they do. However, it could be useful to note that coco.names contains object class names which are all of the discrete classes that this model is capable of detecting.

***Thympi/tests/live\_tests/liveOpenCV.py*** is based on ***object-ident.py*** which contains all of the python code required to set up the model and get started with using the model. The primary change being the ability to easily disable drawing the output image with bounding boxes, which greatly improves detection and classification performance. This is important as it’s a feature that’s used in the final prototype to improve performance.

There are a plethora of options that the user can change when setting up or running the OpenCV model. The following are the ones I believe are most important to know about:

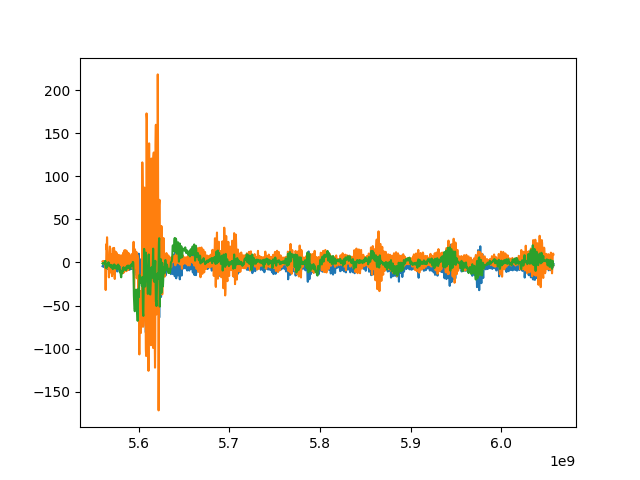
* confidence\_threshold: How confident the model has to be that an object belongs to a certain class in order for it for it to classify it as an instance of such object class
* nms\_threshold: Non-maximum suppression threshold. This determines the threshold which the model will consider a detected object a duplicate. In simple terms, change this if the model is returning duplicate detections.
* draw: Boolean to set for whether the code should just return objects in text form or in image form with drawn bounding boxes (*this only exists in* ***liveOpenCV.py***)

## 

## Stage 5: Live Testing

In earlier stages of implementation, the IMU data was read for the test duration then analysed and displayed at the end of the program as seen in *Stage 3: The MPU6050*. Additionally the Thymio was preprogrammed with certain commands to execute at runtime as seen in *Stage 1: Pi Camera and Thymio Interface.*

Combining these into ***ThymPi/tests/unit\_tests/preprogrammed.py*** allowed me to gather testing data by preprogramming the Thymio with certain commands and then viewing the resultant IMU data. The figure below shows that the raw IMU data looks like after a preprogrammed collision test is performed with a large spike clearly representing the collision event.



##### Figure 3: Raw IMU data after preprogrammed test

After one of the weekly supervisor meetings, we concluded that it would be a good stepping stone to develop a way to read sensor data and control the Thymio in real-time as this would be required for the final prototype.

***ThymPi/tests/live\_tests/liveMPU.py*** uses numpy and matplotlib to visualise and animate the accelerometer data in real-time. Generating a similar output to *Figure 3*except one that updates the output window in real-time as new readings come in. The code for this is loosely based on a tutorial by Shawn Hymel [(*Graph sensor data with python and matplotlib*, no date)](https://paperpile.com/c/0G2eWB/hCXK), however it is quite heavily modified and stripped down for the purposes of this project.

The proposed way to control the Thymio in real time was to bind keyboard events to functions that send corresponding control signals to the Thymio. I knew that this would require some form of event-based loop to be implemented. The first event loop I tried implementing was the pygame game loop which worked on the first attempt allowing me to use keypresses to send control signals to the Thymio in real-time as seen in ***ThymPi/tests/live\_tests/liveThymio.py.*** Combining these 2 files allowed for testing and development to occur much faster as I now had a way to correlate actions to data in real-time.

## Stage 6: Prototyping

### Physical design

The physical design of the system is quite simple with the Raspberry Pi 3B+, power bank, Pi Camera v1.3 and MPU6050 attached to the Thymio robot using tape and double sided tape. A photograph of this can be seen in *Appendix C,* however it should be noted that during testing, the Raspberry Pi wasn’t powered by the mounted power bank but instead by a wall socket.

An interesting thing to note is that the Thymio has a tendency to pitch upwards when it collides or drives into an immovable object. This is notable because it can cause changes in the accelerometer reading as the x-axis (forward to back) will then pick up the reading for gravitational acceleration as the Thymio pitches upwards. The power bank is the heaviest component mounted on the robot, and therefore placing it as far forward would make the system more pitch resistant.

However, another component of compliance testing is the ability for the robot to generate traction. Placing the power bank over the wheels, which are towards the rear of the Thymio, generates notably more traction resulting in the robot being able to push heavier objects.

### Final Code Implementation Breakdown

It was decided that a singleton instance of a **ThymPi** class would be used to represent the robot with instance variables to store learned obstacle compliances and methods to perform the required functionality.

The important functional methods and their brief breakdowns are as follows:

* **testCompliance()** which calls **calibrateSensor()**, performs the compliance test, calls **calcCompliance()** with the IMU data then updates the instance with the compliance data for potential future use.
* **getObjects()** which takes an image as input, uses the OpenCV detection model to detect and classify objects within the image, and returns a list of detected objects and the model’s confidences for them.
* **calibrateSensor()** which calibrates the accelerometer when the system is at rest and returns the mean value and error range for the sensor, which is then used to determine the noise floor.
* **calcCompliance()** which defines how compliance is calculated from the accelerometer data.

Before the main loop of the program can start and the robot can begin testing compliances, some setup needs to be done.

* **setupModel()** needs to set up the OpenCV detection neural network with the pretrained model based on code from ***liveOpenCV.py.***
* **setupThymio()** sets up the interface with the Thymio based on code from ***liveThymio.py****.*
* **setupIMU()** sets up the mpu6050 based on code from ***liveIMU.py***

The main loop of the program has been broken down into extremely high level pseudocode for readability sake. However, it can also be easily found towards the end of ***ThymPi/prod/main.py*** if the reader desires to look into it further.

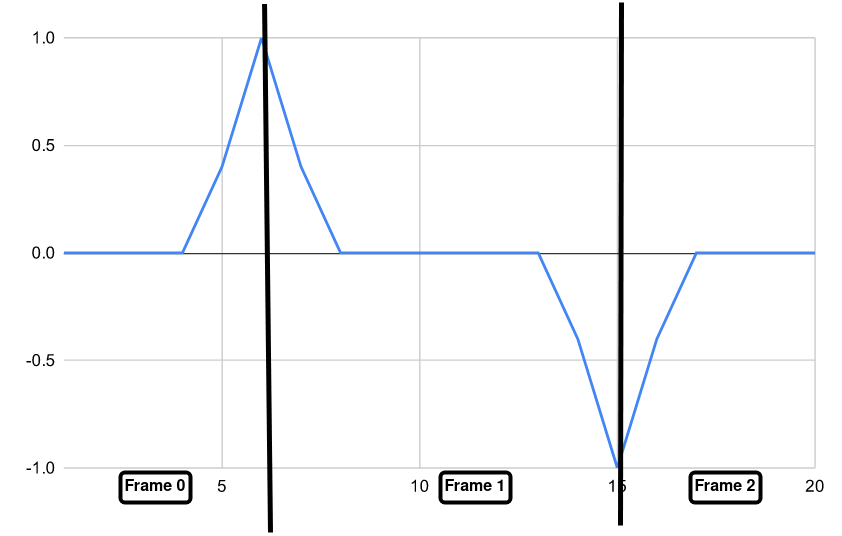
* The main loop continuously checks if there’s an object within the Thymio’s Proximity sensor range of roughly 10cm.
* If there is an obstacle detected, then the robot will go backwards an additional 10cm in order to allow for the camera to get a better view of the object and for the OpenCV detection model to have a better chance at detecting and classifying it.
* Once the obstacle is classified, the main loop checks the singleton instance of **ThymPi** for whether it already knows its compliance.
* In the case that it already knows the obstacle compliance, it will return the obstacle’s known compliance and ask the user if they want to retest the obstacle’s compliance.
* On the other hand, if it does not already know the obstacle’s compliance, it will automatically test its compliance and update the **ThymPi** instance with it.

### 

### Detailed Breakdown of Compliance Testing

The initial approach was more naive, using fixed sized “frames” to break down the raw IMU data into more manageable chunks. The average readings for these frames were then recorded and used to determine if a deceleration event had occurred. Overall, this method actually did initially show some promising results.

The main benefit to using this frame based approach was that it greatly reduced the impact that small fluctuations in the IMU data had. However, the main disadvantage is that this also essentially reduces the sample rate of the IMU. Despite the MPU6050 having a 1000 Hz sample rate, it only averages around 300 readings in real world testing due to overhead from multiple sources. Therefore, when using a frame size of 50, the effective sample rate of the IMU becomes as low as 6Hz, which results in the same low sample rate issues as initially seen in *Stage 2: Exploring Inertial Measurement Units.*



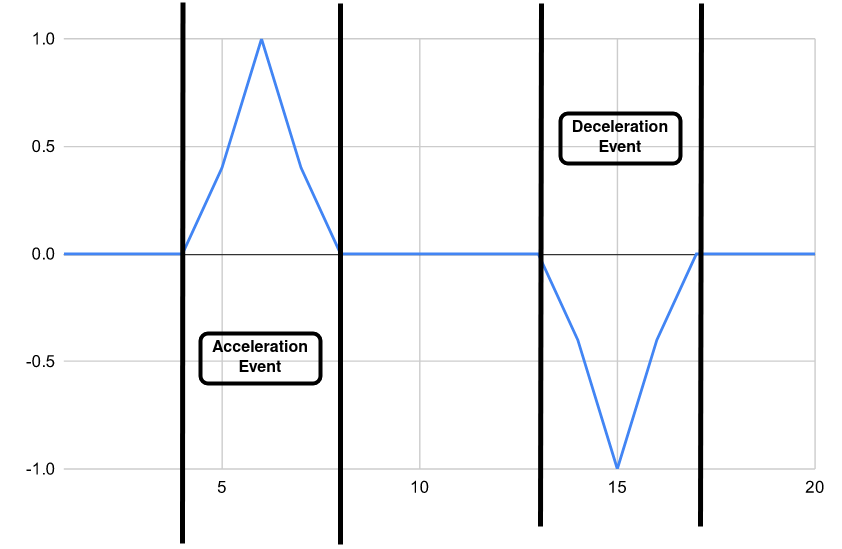
##### Figure 4: A simplified illustration of a frame based approach

*Figure 4* shows a simplified illustration of how a frame based approach can misrepresent the actual data. With frame 1 in the figure having an average of 0 despite having both significant acceleration and deceleration.

A new approach was needed. The idea was to first completely ignore readings that were below a certain noise floor. After calling **calibrateSensor()**, the noise floor is calculated using the following formula.

noise floor = (max value - min value) / 2 - mean value

Once values below the noise floor were ignored, the only values remaining are theoretically either significant accelerations or decelerations. In order to break these down into something more manageable, these are broken down into acceleration and deceleration events which are groups of consecutive acceleration or deceleration events.



##### Figure 5: A simplified illustration of the new approach

*Figure 5* shows how the new approach better represents the actual data. Once the data is broken into acceleration and deceleration events, these events can then be quantified. One rudimentary method to quantify the magnitude of the events is simply multiplying the highest value with the number of readings. Using this, we can assume that the greatest magnitude deceleration event during a compliance test would be when the Thymio collides with the obstacle whose compliance is being tested. This is roughly what is being done by the **calcCompliance()** method.

Once we’ve isolated the greatest magnitude deceleration event that represents the collision event, we can use that to calculate compliance. There are many methods to do this, the one chosen for this project is to use a weighted average. The method **wAvg()** in ***ThymPi/prod/lib/compliance.py*** takes the greatest magnitude deceleration event and sorts the deceleration values in descending order with the greatest deceleration first. It then uses the preprogrammed list of weights to calculate a weighted average. The weights list assigns greater importance to greater deceleration values and vice versa. This can be changed in the future and is by no means the optimal weights list.

# Testing Methodology

The initial proposed testing methodology was to use an indoor testing environment with obstacle analogues in order to simulate an outdoor forest environment with obstacles of varying compliances. The obstacle analogues were to be different coloured cardboard boxes with different physical weights in order to simulate different compliances. It was also assumed that either a pre-programmed route would be used or a simple navigation algorithm would be implemented if time permits.

Once it had been decided that OpenCV would be used for object detection and classification, the decision was then made to modify the testing methodology. The different coloured boxes were now to be replaced with different objects with everything else remaining the same.

Due to time constraints, it was decided that implementing a navigation algorithm would be impractical. A proposed alternative was to have a human operator in place of a navigation algorithm. Where the operator would control and position the robot in the testing environment. However, once the robot encounters an obstacle as detected by the proximity sensor it would either autonomously test it if it’s compliance isn’t known or prompt the operator if they want to retest its compliance.

Meeting this aforementioned expected behaviour would be considered a qualitative pass of the test. While quantitative metrics weren’t the focus of this project, some possible metrics that could be tested include:

* Reliability in edge cases
* Repeatability of measured compliance values
* Ability to detect small changes in compliance

This testing methodology still demonstrates the project goals of autonomously testing obstacle compliance using sensor data and using it to make future decisions. Additionally, it would also be quite easy to modify and extend the project and replace the human operator with a navigation algorithm in the future. This possibility will be further discussed in the *Conclusion* section.

# Conclusion

The prototype was able to successfully meet the project goals as set in the *Introduction* section of this report by reliably autonomously determining obstacle compliance using sensor data as tested using the final revised testing methodology. However, it should be noted that there may need to be certain future tweaks to the **wAvg()** method in order to better calculate compliance based on sensor data as the current approach certainly isn’t optimal, but is simply a working proof of concept.

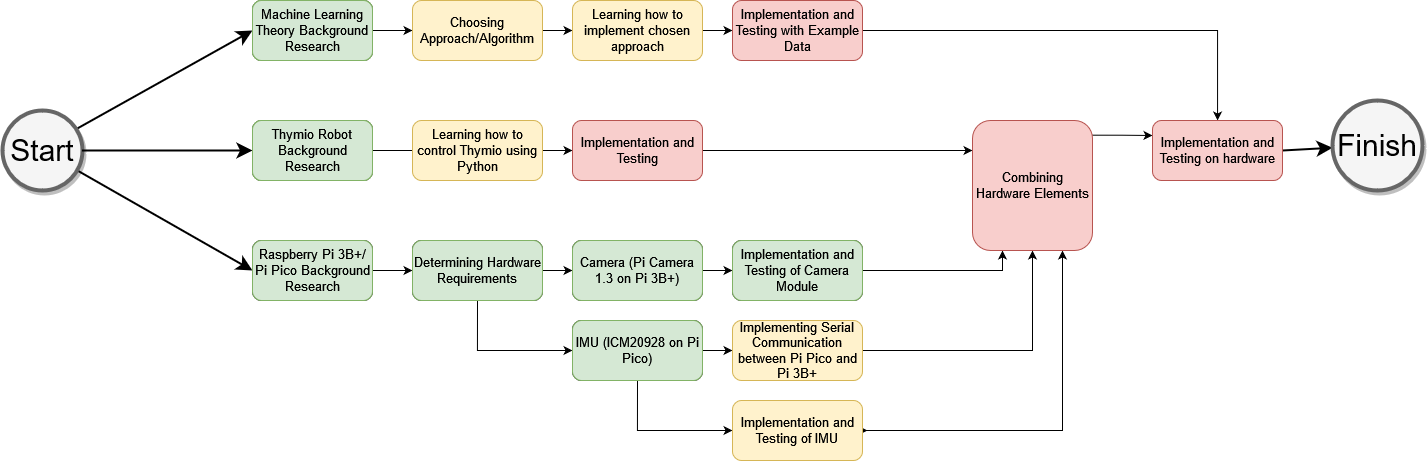
Additionally, we also met the goals set out in the Initial Project Brief (available as *Appendix D*) to a high degree. With the notable exception being that we didn’t end up pursuing anything involving improving overall navigation efficiency.

The first notable change in scope of the project was the change from using obstacle colour to classify obstacles but instead using an OpenCV detection and classification model. This can be considered an increase in scope as it added additional functionality to the system. It also has the added benefit of being more similar to a potential real world deployment. This change is detailed further in *Stage 4: OpenCV.*

Another notable change in scope of the project was the omission of any navigation algorithm of preprogrammed pathing due to time constraints in favour of having a human operator in place of one. This change ended up not affecting the actual goals of the project and is further explained in the *Testing Methodology* section.

The most difficult part of this project was working with IMUs. First having to completely abandon the approach of ICM20928 IMU on the Pico4ML as the IMU for the project despite significant time and effort invested into it due to interfacing unreliability. Then having to abandon the Thymio’s onboard accelerometer due to it having an insufficient sample rate and precision for this project . Eventually having to resort to purchasing the MPU6050 IMU instead. This process is explained in detail in *Stage 2: Exploring Inertial Measurement Units* and *Stage 3: The MPU6050*.

Additionally, Looking back at the PERT chart from the progress report gives us a framework to compare the final prototype to the planned completion point at an earlier stage of the project. Additionally, it also provides a good segue into discussing the changes to the implementation and testing that were made throughout the process.



##### Figure 6: PERT chart from progress report

The most significant change was that it was initially believed that I would use some form of linear regression to correlate obstacle colour with obstacle compliance. Alongside this, the possibility of implementing confidence intervals for this was also entertained at the time. However, this was eventually changed to a more discrete approach with obstacle colour and obstacle compliance instead forming a key-value pair.

Additionally, it was completely unexpected that there would be a significant intermediary stage of development where live testing would be conducted without necessarily having a full or even partially complete prototype. It was assumed that a prototype would be physically built and development would commence from there as represented by the large “Combining Hardware Elements” and “Implementation and Testing on hardware” components of the PERT chart in *Figure 6*. The explanations for why live testing was a significant stage of this project’s development are explained in further detail in *Stage 5: Live Testing*, whereas the actual prototyping is explained in its own section in*Stage 6: Prototyping*.

## Potential Future Developments

As mentioned at the start of this conclusion, the current implementation of using a weighted average wAvg() function to calculate compliance is far from an optimal solution. There could possibly be further research and testing into collision physics and how to actually quantify deceleration in order to calculate obstacle compliance. This would likely be the most obvious and significant improvement to the current project.

Additionally, a multi-sensor approach as proposed in the *Analysis and Design*section would also be a significant improvement over the current single sensor approach to determining obstacle compliance and allow for significantly greater precision and reliability.

Apart from just improvements to the current project, the concept of testing compliance using on board sensors to supplement an autonomous vision-based navigation system in an environment with compliant obstacles has further applications. By demonstrating that accelerometer data can be used to determine obstacle compliance, a future project involving autonomous navigation of environments with compliant obstacles could then use an accelerometer or other sensors alongside the learnings of this project to supplement their navigation algorithm.

# References:

[*Arducam Pico4ML TinyML Dev kit: RP2040 board w/ QVGA camera, LCD screen, onboard audio, reset button & more* (2021) *Arducam*. Available at:](http://paperpile.com/b/0G2eWB/9B0k) <https://www.arducam.com/product/rp2040-based-arducam-pico4ml-dev-board-for-machine-vision/> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/9B0k)

[Brako, M. (2022) *Vision-Based Navigation for an Autonomous Rover in a Forest Environment*. Edited by D. Tarapore. MEng Electronic Engineering with Artificial Intelligence. University of Southampton.](http://paperpile.com/b/0G2eWB/G8Qm)

[*Graph sensor data with python and matplotlib* (no date). Available at:](http://paperpile.com/b/0G2eWB/hCXK) <https://learn.sparkfun.com/tutorials/graph-sensor-data-with-python-and-matplotlib/update-a-graph-in-real-time> [(Accessed: 3 May 2023).](http://paperpile.com/b/0G2eWB/hCXK)

[Hofer, B. (no date) *thympi: Control thymio over dbus bindings and asebamedulla with python, running on a raspberry pi*. Github. Available at:](http://paperpile.com/b/0G2eWB/8Fzf) <https://github.com/lebalz/thympi> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/8Fzf)

[*Home* (2021) *Thymio*. Thymio by Mobsya. Available at:](http://paperpile.com/b/0G2eWB/yLt2) <https://www.thymio.org/> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/yLt2)

[Koetsier, J. (2023) ‘Self-Driving Investment Crash: 58% Drop In Autonomous Vehicle Dollars’, *Forbes Magazine*, 19 April. Available at:](http://paperpile.com/b/0G2eWB/cH3e) <https://www.forbes.com/sites/johnkoetsier/2023/04/19/self-driving-investment-crash-58-drop-in-autonomous-vehicle-dollars/> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/cH3e)

[Martijn (no date) *mpu6050: A Python module for accessing the MPU-6050 digital accelerometer and gyroscope on a Raspberry Pi*. Github. Available at:](http://paperpile.com/b/0G2eWB/uBnR) <https://github.com/m-rtijn/mpu6050> [(Accessed: 3 May 2023).](http://paperpile.com/b/0G2eWB/uBnR)

[Newlands, C. (2021) *Generating and Rendering Realistic, Navigable Forest Environments*. Edited by K.-P. Zauner. MEng Computer Science. University of Southampton.](http://paperpile.com/b/0G2eWB/EtNl)

[*Programming interface - thymio & aseba* (no date). Available at:](http://paperpile.com/b/0G2eWB/BV7J) <http://wiki.thymio.org/en:thymioapi> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/BV7J)

[*Raspberry Pi Camera V1.3 —* (no date) *Maker Portal*. Available at:](http://paperpile.com/b/0G2eWB/fldY) <https://makersportal.com/shop/raspberry-pi-camera> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/fldY)

[Sina (no date) *sensor\_readings at master · SinaSarparast/Thymio*. Github. Available at:](http://paperpile.com/b/0G2eWB/k5FW) <https://github.com/SinaSarparast/Thymio> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/k5FW)

[*The ARGoS Website* (no date). Available at:](http://paperpile.com/b/0G2eWB/oq7y) <https://www.argos-sim.info/> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/oq7y)

[*thymio: Thymio simulator based on ARGoS framework* (no date). Github. Available at:](http://paperpile.com/b/0G2eWB/DT02) <https://github.com/resilient-swarms/thymio> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/DT02)

[Tim (2021) *Object and Animal Recognition With Raspberry Pi and OpenCV*, *Core Electronics*. Available at:](http://paperpile.com/b/0G2eWB/7wWy) <https://core-electronics.com.au/guides/object-identify-raspberry-pi/> [(Accessed: 2 May 2023).](http://paperpile.com/b/0G2eWB/7wWy)

# Appendices

## Appendix A: GitHub Repository for ThymPi Project

The project source code is available on GitHub at ***https://github.com/yaq1n0/ThymPi*** as minor changes may be made to it over time.

## Appendix B: Hardware Costs

| **Item:** | **Price** | **Notes:** |
| --- | --- | --- |
| Thymio Robot | N/A | loaned from the university |
| Raspberry Pi 3B+ | N/A | already owned |
| Arducam Pico4ML | N/A | already owned,  not used in final implementation |
| 5000mAh Power Bank | £19.99 | purchased from Amazon |
| Raspberry Pi Camera v1.3 | £23.97 | purchased from Amazon,  item count=3 |
| MPU6050 IMU | £11.99 | purchased from Amazon,  item count=5 |
| Misc wires/cables | N/A | already owned |

## Appendix C: Photograph of Testing Setup



## 

## Appendix D: Initial Project Brief

**Problem Statement:**

When navigating outdoor environments, it is important to note that different

obstacles tend to have different levels of compliance. Compliance in this context

being defined as whether the obstacle can be moved or otherwise manipulated.

For example, in a forest, a robot navigating the environment must understand that

grass and foliage are compliant and can be navigated through but rocks and logs

above a certain size are non-compliant and must be navigated around.

An approach involving the robot physically interacting with the environment and

testing each obstacle’s compliance would be effective. However, there are

drawbacks to performing a physical test of an obstacle’s compliance every time the

robot encounters one.

**Goals:**

In this project, we would like to determine if it is possible to get a robot to learn

what obstacles are compliant once and then apply this understanding when it

encounters similar obstacles in the future. Ideally this would be achieved using

purely visual means through use of simple object recognition. This could potentially

make the robot more time and energy efficient at navigating these environments.

**Scope:**

This project is part of a larger forest navigation problem, focusing on

intelligently identifying compliant obstacles and therefore we will be abstracting

certain parts of this project. We will use an indoor environment with obstacle

analogues to simulate an outdoor environment for testing purposes. We will also be

using a simpler robot for proof of concept.

A possible extension we might also explore is how the robot might be able to

make intelligent decisions regarding the most time and energy efficient route to take

within an environment with compliant obstacles. For example, it can decide to take

the longer route with shorter grass rather than the shorter one through dense but navigable foliage.