TÉCNICO LISBOA

Drones for Intralogistics using IoT

Guilherme Bacharel Ivens Ferraz Portela

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Prof. Susana Isabel Carvalho Relvas

Examination Committee

Chairperson: Prof. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Member of the Committee: Prof. Miguel Filipe Leitão Pardal

January 2021

Acknowledgements

Este projecto não teria sido possível sem o apoio que tive. Agradeço à Mãe, ao Pai, ao Gonçalo, à Filipa (e ao Iron) pelo apoio diário e pelas palavras sábias. Ao João Santos e à Sara Franco, ao Guilherme Morais, ao João Lopes, ao André Gil e ao Francisco Lopes por me aturarem. À Professora Teresa Vazão, à Professora Susana Relvas e ao Professor Pedro Diniz pela excelente ajuda e disponibilidade na orientação deste projecto em tempos complicados. Ao IST por me facultar a possibilidade de aprender tudo o que precisei para chegar ao fim deste percurso e pelo espaço e equipamento necessário prestado.

Abstract

There is a growing desire for warehouse automation to reduce operational costs and improve logistics efficiency. In this work, we survey the state-of-the-art on the use of indoor drones for warehouse inventory leveraging current Radio Frequency Identification (RFID) technology. We propose a system architecture to automatically carry out warehouse inventorying by detecting RFID item tags while autonomously navigating the facility. A routing algorithm was developed for a Reserve Area of configurable dimensions. The navigation is complemented by leveraging OpenCV and the frontal camera to perform mid-flight corrections based on visual markers serving as reference points. We outline the proposed system architecture and develop it as a proof-of-concept design.

Keywords

Drone, Inventory Management, RFID, Internet of Things (IoT), Computer Vision, Python.

Resumo

Existe uma procura cada vez maior para a automatização de armazéns com o objectivo de reduzir custos e melhorar a eficiência da cadeia de logística. Este trabalho explora o estado da arte em termos da utilização de drones em espaços interiores para inventariar armazéns através da utilização de tecnologia RFID e propõe um sistema para inventariar armazéns automaticamente. Este sistema utiliza tecnologia RFID para detectar etiquetas colocadas em iténs enquanto percorre o armazém autonomamente. Foi desenvolvido um sistema de planeamento de rota para uma Área de Reserva de dimensões configuráveis. Este sistema é complementado pela utilização de OpenCV em conjunto com a câmera frontal para fazer ajustes a meio do voo do drone consoante marcas visuais que são utilizadas como pontos de referência. É elaborada uma proposta de implementação para este sistema e é desenvolvida uma versão para prova de conceito.

Palavras Chave

Drone, Gestão de Inventário, RFID, IoT, Visão Computacional, Python.

Contents

List of Figures

Acronyms

IoT	Internet of Things
RFID	Radio Frequency Identification
UAV	Unmanned Aerial Vehicle
SOAP	Size, Orientation, Angle, Placement
UWB	Ultra-Wide Band
SBCS	Shortcut Bissected Countdown Scheme
LAN	Local Area Network
GPS	Global Positioning System
RSSI	Received Signal Strength Indicator
SBC	Single Board Computer
EPC	Electronic Product Code
SDK	Software Development Kit
VPS	Vision Positioning System
IMU	Inertial Measurement Unit
USB	Universal Serial Bus
FTP	File Transfer Protocol
UHF	Ultra High Frequency
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1

Introduction

Contents

Introduction

1.1 Purpose and motivation

Recent developments in sensor technology have fueled a revolution in what is commonly named the IoT. Inexpensive and advanced sensors coupled with sophisticated communication and air-borne platforms (e.g. drones) have enabled the development of extremely versatile distributed sensing and computing solutions. These systems cover a wide range of application domains, such as rapid re-forestation [1], crop planting and treatment [2], and last-mile aerial deliveries [3], just to name a few. When considering warehouse inventorying of available stock, it is a very repetitive and time-consuming process that requires laborious and error-prone manual counting. This makes it increasingly desirable to have flexible and accurate automation. Similarly to the application of drones for last-mile deliveries (which can increase the agility of a delivery system when reaching the end-consumer), another possible application of particular interest is the use of these technologies for near real-time warehouse inventorying, key in modern-day supply chains. Knowing which products can be located where amidst the shelves allows for faster dispatch and delivery, reduced time spent in shelves for products which leads to an increase in operation efficiency. This increase in efficiency is complemented by additional services, such as next-day delivery, adding value (and hence increasing profit margins) to the supply chain between producers and consumers.

Warehouse management [1] is a prime candidate for advanced solutions using IoT technologies and drones. Items can be catalogued with RFID tags, enabling a great deal of information to be stored and thus enabling easy tracking and counting [2]. In addition, certain drone types are small enough to be operated indoors, allowing for the automation of tasks traditionally performed by humans [3]. In terms of inventorying large warehouses, drones can more easily travel to hard-to-reach areas (far away, high up, across obstacles), simplifying the time-consuming, error-prone and potentially hazardous task of manually checking the stock in a given warehouse [4]. Besides the essential storage system and individual units (such as pallets or boxes), the personnel and their equipment, the warehouse is outfitted with an

[1] DroneSeed Rapid Reforestation Drones: https://www.droneseed.com/
[2] Detailed use of drones in agriculture: https://uavcoach.com/agricultural-drones/
[3] Amazon Prime Air: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011

up-to-date product database and specialized order-picking equipment. It is important to maintain a high degree of item identification accuracy and to validate an item's storage location inside the warehouse. In particular, it is of interest to further explore how to improve the storing and order-picking process and how to streamline updates to the database.

Automating this set of tasks is far from trivial, however. In general, a warehouse's operation is subdivided into four basic functions: receiving (items are unloaded and put away), storing (either in the more economical Reserve Area or in the quicker and easier to access Forward Area), order-picking (fulfillment of orders by item retrieval) and shipping (items are loaded and shipped) [5]. Warehouse management optimization presupposes that these processes are properly defined [6].

After an item arrives, it is stored according to specifications which vary between warehouses. The storage process then allows for the item to be found and collected during order-picking which assumes a properly inventoried stock (e.g. the available quantity of an item and its current position must be known). The human labour typically used is naturally prone to mistakes, particularly in this scenario in regards to the storing of items and their order-picking. If the item is misplaced or in insufficient quantity, the order-picking process will fail to meet the expected results. To guarantee an accurate reading of stock, arrival and shipping operations may have to stop for a few days[4].

To summarize, current drone technology can be leveraged to provide accurate, near real-time updates to item counts and positions across a warehouse quickly while ensuring warehouse operations continue in a safe and controlled manner. Further study on warehouse-oriented drone applications is an interesting prospect to increase the efficiency of current supply chain management solutions.

1.2 Goals and contributions

The objective of this thesis is to propose solutions for the automation of the process of inventory management in a warehouse.

The contributions for this goal are as follows:

- System prototype for automatic warehouse management based on a drone as a replacement for human labour (in the identification, counting and validation of items across a warehouse). The developed code throughout this project is hosted on Github [7];
- Automated use of RFID technology for cataloguing shelved items or item storage units;
- Indoor warehouse navigation based on a flexible virtual coordinates map, computer vision and RFID technology.

[4] L'Oréal & the inventory drone Eyesee: https://youtu.be/HgNWoSFEwv4, 02 Jan 2020

1.3 System overview

The proposed solution entails a drone-based computational platform that queries surrounding passive RFID tags to count and identify misplaced items, and to update the warehouse's database. The drone also navigates the warehouse's indoor storage area in an efficient and autonomous fashion, correcting its course whenever needed. The developed software consists of three components: The inventorying component, tasked with updating the local database; the navigation component, tasked with routing, assigning movement commands to the drone and correcting its course; the communications component, which provides a user interface, updates the warehouse database and allows for the initialization of the warehouse inventorying software.

1.4 Document organization

This report is structured into four major sections. In section 2 (State of the Art) we describe the following aspects of the work. The use of drones for intralogistics and inventory management; the sensors used in warehouses; the current methods for remotely updating a warehouse's database; the current navigation and positioning methods; the constraints (both of the physical world and legal boundaries); related work on drone-based warehouse management solutions. In section 3 (System Architecture) we describe a proposed system architecture for the use of drones for inventory management. Here, we focus on the gathered information in the State of the Art section; the analysis of the system requirements according to the assumptions; the system architecture, in overview and specific software components. Section 4 (Warehouse Inventorying System Implementation) is discusses the development process of the project. Each subsection details the choice of materials, their use and implementation and the setbacks experienced when working with them. In section 5 (Testing and Results) we present the results of testing each part of the project, discuss them and consolidate the most important points of the proposed work. Finally, section 6 (Conclusions) addresses the conclusions, presents the future prospects of this field and sets out goals to build toward with this work as a base.

2

State of the Art

Contents

State of the Art

This chapter describes the state of the art regarding drones for intralogistics. In section 2.1, we survey different types of drones and discuss their suitability for inventory management. In section 2.2, we survey sensors suitable for being drone-mounted and placed on warehouse inventory. In section 2.3, we survey inter-drone and drone-to-base communication technology. In section 2.4, we survey algorithms and protocols used for the positioning of the drone. In section 2.5, we outline physical and legal limitations of drone use for inventory management and describe related use cases in section 2.6. Lastly, section 2.7 presents a summary of the topics described during this section. Throughout this section, a list of assumptions about the warehouses and their inner workings can be defined in order to fully encompass the system architecture to be developed in section 3.

2.1 Drones for Intralogistics

Unmanned Aerial Vehicle (UAV)s, usually referred to as drones, serve a variety of purposes and are beginning to play an important role in logistics. Before diving into the technical aspects of the use of drones in intralogistics, it is important to get a brief overview of existing drone types.

2.1.1 Drone Types

Drone form-factors range between Fixed-wing, Unmanned Helicopters, Multicopters, and Tilt-wing.[1] Fixed-wing drones are analogous to regular aeroplanes as they share the same mechanism to produce lift - forcing airflow around a fixed (but possibly with variable geometry) wing. They are generally very efficient as they only require energy to move forward. Fixed-wing drones, however, cannot hover in one spot but can instead maintain motion for extended periods of time, thus enabling them to cover large distances. They tend to need ample berth to be launched (catapult or runway) and retrieved (parachute, net or runway). Their characteristics are leveraged in their use, such as for aerial mapping/surveying or even in the inspection of widespread infrastructures such as long pipelines, power lines or roads.

[1] Drone types: https://www.auav.com.au/articles/drone-types/, 27 Dec 2019

Multicopters (or multi-rotor drones) are the most common type of drones, due to their compact nature, ease of use and comparatively accessible price point. They have limited load capacity and speed, making them reliable only for quick tasks and/or short operational ranges. Multicopters have the ability to hover, but the energy cost to generate lift is high. Nevertheless, their affordability and reduced sizes allow for applications in confined spaces and indoors. Their typical uses vary from photography to video-based aerial inspection and even to small parcel delivery (e.g., Last Mile Delivery system offered by Amazon's Prime Air [8] greatly cuts costs for fuel and allows for both a much larger and quicker coverage of delivery goals [9]). Unmanned helicopters, electrically or gas powered, can hover very efficiently given their large single blade coupled with the tail rotor for heading control. Their relative large size, complexity and inherent danger near people makes their use-case limited to particular situations, mostly outdoors. Still, they find use in applications such as precision agriculture[2] at a scale unfeasible for smaller multicopters. Tilt-wing or Fixed-wing hybrid drones provide a compromise between hovering and propeller-based movement. The technology that supports them is still complex and expensive, yet they offer an interesting middle ground between multicopter and fixed-wing drones in terms of uses.

2.1.2 Drones for intralogistics operations

Considering the work's objective in warehouse intralogistics, the multicopter represents the best choice due to its size, ease of use and price point. They tend to lack payload capacity and are sensitive to wind speed changes[3]. Still, these weaknesses are mitigated by the gradual development of drone technology and the fact that most warehouses are indoors. Since multicopter drones are already being used for last-mile aerial deliveries, they can also be used as automated order-pickers in warehouses. Intralogistics-oriented drones can be used to carry out inventorying tasks with minimal disruption in regular warehouse working conditions.

2.1.3 Synthesis

In summary, multicopter drones are the most adequate UAVs for warehouse settings as their characteristics make them the best choice for tasks such as inventorying stock and light order-picking. Even in outdoor settings, they are already used for light-weight last-mile deliveries. As the technology behind multicopters matures, thus making them more efficient (allowing for larger payload and longer flights), we expect their use to become even more widespread for a wider variety of supply chain management tasks.

[2] R22-UV unmanned helicopter joins UAVOS fleet for precision agriculture:
https://www.therobotreport.com/r22-uv-unmanned-helicopter-uavos-precision-agriculture/, 27 Dec 2019

[3] Unmanned Aerial Vehicle in Logistics:
https://www.dhl.com/content/dam/downloads/g0/about_us/logistics_insights/DHL_TrendReport_UAV.pdf, 07 Dec 2019

2.2 Warehouse Sensors

The three main sensing methods currently employed by drones for identifying, counting and checking the state of an item in a warehouse are, barcode scanners, optical sensors, and RFID readers.

2.2.1 Barcode Scanners

Barcode readers consist of a light source, a lens and a light sensor translating for optical impulses into electrical signals. They are capable of reading a printed barcode, decoding its data and sending the data to a computer. This scanning technology has been used both manually and automatically for some time as a reliable means to properly manage the supply chain [2]. This identification technology is considered to have matured, as barcodes are simple to generate and read, expected to remain minimal in terms of costs and currently operate as an everyday item. The referred accuracy values for this method's are greater than 99.99% [2]. Drones can be easily equipped with barcode scanners to perform routine warehouse inventories. Barcode scanning has a few drawbacks, namely the fact that the scanner requires the object to be inventoried to be in its line-of-sight. The scanner has no object-penetration capability, and can therefore only scan the frontal layer of items in a shelf, for example. Due to the properties of the scanner, the drone would most likely need to traverse each shelf in both the horizontal and vertical axes. This problem is present in both one-dimensional (linear) barcodes and two-dimensional barcodes (such as QR codes).

2.2.2 Optical Sensors

Camera and video technology is currently used for a variety of warehouse applications, including in support of drone movement and visual identification of stock.

Computer vision can be used on a live feed from the drone's camera to supplement its movement as a feedback system. By using reference points such as a coloured line along the floor or markings on walls, the computer vision module can use them as references to correct the drone's movement or to send a warning if the drone moves off-course.

In some contexts, image recognition software is coupled with live recording for real-time identification of warehouse items through one-dimensional barcodes [10]. This however, presents drawbacks similar to the ones in barcode scanning as line-of-sight is once again a requirement and adequate lighting conditions are needed to ensure accurate image recognition performance.

The leveraging of the drone's optical sensors is an interesting option as it provides a strong link to the real world for a drone system to interpret. The sensors' dependence on proper lighting can be solved by attaching a dedicated light source to the drone.

2.2.3 RFID Sensors

RFID relies on electromagnetic waves in the radio frequency band. Functionally, an RFID system consists of two separate parts: the tag and the reader.

The RFID tag, typically consisting of a microchip, is capable of storing information such as a product identifier (like a serial number) and is connected to an antenna [2]. In order to maximize read range and signal quality, tags should be placed according to the Size, Orientation, Angle, Placement (SOAP) factors[4]. These tags can also be classified as "passive" or "active" (see Figure 2.1 for clarification).

Passive RFID tags respond to a specific frequency with information they have been previously recorded with upon being energized via the process of electromagnetic coupling by the RFID reader. They thus require a specific distance and preferably unobstructed line-of-sight to a reader to operate [11]. These tags are subdivided into the frequency bands they operate in, from Low Frequency (125 kHz to 135 kHz), to High Frequency (around 13.56 MHz), to Ultra-High Frequency (between 868 MHz and 926MHz) even up to Microwaves (at 2.45 GHz and 5.8 GHz).

Active RFID tags are battery powered and can keep broadcasting their signal without stimulation from the RFID reader. While passive RFID have standard specification regarding their frequency responses, active tags are developed and employed in specific applications and are not yet standardized [12].

The second component of the system is the "RFID Reader", which is needed to interact with the RFID tags. The reader can read or write to these RFID tags through a query signal it emits [2]. An important consideration for this system is that the reader does not require a line-of-sight to the tags, as the electromagnetic waves can penetrate (up to a specific frequency dependent depth) solid objects. The antenna on a reader may operate in two different sections of the electromagnetic field that surrounds it: the near-field and the far-field. The diameter of the antenna compared to the half the wavelength of the radiation it emits impacts the expression used for defining the boundaries of each section of the field. The near-field consists of the area surrounding the antenna up to one wavelength and is mostly magnetic in its operation. Through inductive coupling, the antenna energizes the tags with a magnetic field and decodes the disturbances the tag responses cause in it. The far-field consists of the field beyond the near-field and has both magnetic and electric components. By using capacitative coupling, the reader emits Radio Frequency energy to energize a tag, which uses the received energy to respond. This response is referred to as backscatter[5].

RFID readers can query the tags by signalling (or querying) one particular tag at a time or engaging in bulk-reading where multiple tags are simultaneously queried. Both cases are subject to collisions. The bulk reading method is particularly interesting from a logistical point of view, as its use with the Shortcut Bissected Countdown Scheme (SBCS) protocol (as opposed to the typically used Query Tree

[4] RFID Tags: How to Get the Best Read Range: https://youtu.be/D-AOOoSFj78, 09 Dec 2019

[5] Understanding Near-Field and Far-Field Antennas: https://www.atlasrfidstore.com/rfid-insider/near-field-vs-far-field-rfid-antennas, 21 Set 2020

[13]), coupled with more than one antenna [14], has an accuracy comparable with that of the individual barcode scanning method.

2.2.4 Synthesis

In conclusion, a low-maintenance and mature sensor system is an ideal solution for warehouse sensorization. Considering the technologies above, RFID may provide the most interactive and interoperable sensor system.

Sensor System	Power Consumption	Line-of-sight	Range	Cost
Barcode Scan	Scanner must be powered	Required	Below 1m	Very low (readers, printed barcodes)
Optical Scan	Optical sensors and extra lighting must be powered	Required	Dependent on camera quality and lighting	Medium (cameras, improved lighting, software, maintenance)
RFID Scan (passive)	Reader must be powered	Not required	Dependent on near-field or far-field operation	Low (readers, cheap tags)
RFID Scan (active)	Reader and tags must be powered	Not required	Between 50m and 2000m	High (readers, expensive individual tags, maintenance)

Figure 2.1: Comparison between available warehouse sensors.

2.3 Warehouse Database Updates

While drones use sensors to accurately and consistently read the requested information from their surroundings, they must also communicate with the warehouse's database for updating the state of the warehouse in terms of location and quantity of the items. We examine this drone-infrastructure interaction, in two aspects, namely, transmitting information to the database; the processing of received information at the database.

2.3.1 Transmitting Information

A simple communication solution would consist in manually transferring the drone information (after it performed its routine inventorying) into the warehouse's database system. A human could manually connect the drone to an interface computer from which the warehouse database would be updated subsequently setting the drone up for its next routine inventory check. This approach, however, does not leverage the advantage of the drone's coverage and range as it would require it to return to the recovery

point after each routine inventory check. Furthermore, it does not guarantee the real-time update of information, as by the time the drone completes its run, additional items may have left and/or entered the warehouse.

Consequently, solutions to be considered would preferably be wireless to leverage the drone's reach, its reliability and opportunity for real-time communication. A typical commercial-grade drone communicates with the its controller through radio frequencies in the 2.4 GHz and 5.8 GHz spectrum bands.[6] Although a Wi-Fi-based solution might seem desirable, it is important to consider the spatial context of the warehouse setting, as they vary in terms of size reaching hundreds of meters in length. It has been shown [15] that the Wi-Fi protocol is ill-suited for indoors featuring several layers of obstacles between a stationary control unit and a flying drone. Its theoretical maximum range and throughput are hindered by the metallic interference of obstacles such as shelves and inventory as well as the issue of multi-path signal propagation. A solution to consider would include the use of several Wi-Fi access points for complete warehouse coverage. A computational module such as a Raspberry Pi with Wi-Fi capabilities can be mounted on a drone for near real-time data transmission.

An RFID solution can also be considered, as described in the work by the Massachusetts Institute of Technology's RFly Project [11]. In this approach drones act as relays for incoming queries by a centralized RFID reader and can act as hubs for the returning responses. This approach has several advantages. First, the drone is not hindered by the increased weight of an RFID reader, instead having mounted on it a relay (signal booster/repeater). Second, such a relay system greatly increases the coverage of the query made by the RFID reader. Third, the system can be further developed into a scalable version, allowing for several RFID readers to be placed in strategic locations, or a daisy-chain of relay drones communicating with a centralized RFID reader. While conceptually elegant, this approach still needs to address some technical challenges such as the backscatter of an RFID query on a multitude of passive RFID tags. A technical approach to deal with these issues is described in [11].

2.3.2 Processing Information

A modern warehouse is typically equipped with a database system for a variety of operational functions, including the integration of data collected by drones in a near real-time fashion, possibly using the system's wireless infrastructure. This is the case of the work described in [16] in which the authors developed a proprietary software to interface with the drone[7].

[6] The 2.4GHz band is used for the Wireless Local Area Network (LAN) channels for the 802.11b/g/n/ax protocols. The 5.8GHz band allows for higher transmission speeds at the cost of range, making it less useful in the context of inventory management: small amounts of information per item (like the product code) need to be transmitted over a significant distance.

[7] It is, however, unclear from this description if in this solution data is transferred in a wired or wireless fashion.

2.4 Drone Navigation

We now look into alternative drone positioning systems that can support a drone's navigation system in an indoors facility.

2.4.1 Global Positioning System

Global Positioning System (GPS) is the most widespread positioning system worldwide. It relies on a Global Navigation Satellite System capable of geolocating a user as long as at least 4 satellites are in line-of-sight of the user. The system has global coverage and, can have an error between 30 and 500 centimeters [8]. It is clear that GPS would only be viable for an outdoor storage facility, as a roofed warehouse would not allow GPS to operate correctly due to the lack of line-of-sight between the drone and the satellites.

2.4.2 Received Signal Strength Indicator

The Received Signal Strength Indicator (RSSI) value allows for the measurement of a radio signal's power [17] thus enabling the inference of a drone's position by computing the signal strength between multiple known signal sources and the drone. While RSSI values can be obtained from several different established standards such as Wi-Fi, Bluetooth and RFID, they are relative and vary between chip manufacturers. RSSI measurements are also sensitive to multi-path, fading, non-line-of-sight measurements and diffraction [18]. Still, and despite some inherent problems using RSSI as a basis for positioning, it is possible to use it in the context of the following communication infrastructure technologies:

- **Wi-Fi –** Triangulation of a drone's position can be accomplished by using several Wi-Fi access points acting as beacons or markers. The work described in [18], reports a 2.3 meter error for line-of-sight scenarios and a 2.9 meter margin of error otherwise.

- **Bluetooth –** A similar triangulation approach can be use relying on Bluetooth. The work described in [19] reports a location error of 1.7 meters based on a non-moving device, while the work described in [20] reveals that this type of positioning system can work for triangulating a user's position, but for small and decluttered areas (6x8 square meter). It is thus safe to assume that for larger areas with obstacles and drones moving relatively fast both horizontally and vertically, this RSSI-based method would perform worse.

- **RFID –** Direct measurement of signal strength using RFID responses is also possible as shown by the work described in [21]. By applying sophisticated mathematical techniques (e.g. Kalman

[8] GPS Accuracy: https://www.gps.gov/systems/gps/performance/accuracy/, 09 Dec 2019

filtering) to the raw RSSI information, a relatively high accuracy (around 10 centimeter for the absolute positioning error) at a 2 meter reader-tag distance can be achieved. This approach can be improved by leveraging the tag's read rate and response times.[9]

2.4.3 Ultra Wide Band pulse-based positioning

Ultra-Wide Band (UWB) positioning relies on the interval between transmitted and received pulses in the 3 to 5GHz range. Reflections are easily filtered thus mitigating multi-path issues while not interfering with other used Radio Frequency signal bands. In addition to achieving very low position errors (in the order of the centimeters) it also does not require line-of-sight operation.[10] Work in the literature [4] report, for ranges of 6 meters, positioning errors below 15 and 60 centimeters for line-of-sight and non line-of-sight scenarios, respectively. The project described in [4], based on a battery-powered backbone of anchors for indoors positioning, achieves positioning errors below 20 centimeters for up to 75 meter distances with a relatively low power consumption per anchor required (3.4 mA on stand-by and 26.6 mA when used for ranging). The system is scalable, as the maximum communication range grows with the number of anchors installed across an expanded area. These features make the UWB anchor positioning method very suited as a warehouse drone positioning system.

Positioning System	Range	Accuracy (error)	Line-of-sight	Cost	Robustness
GPS	Extremely long	< 5m under open sky	Required (outdoors-use only)	Minimal (widespread infrastructure)	Medium (signal blockage and reflection issues)
RSSI Wi-Fi	Wi-Fi signal range	LoS: <2.3m, NLoS: <2.9m	Not required	Medium Wi-Fi Acess Point price	Low (multipath, fading, diffraction issues)
RSSI Bluetooth	Bluetooth signal range	1.7m with LoS	Not required	Low Bluetooth device price	Low (multipath, fading, diffraction issues)
RSSI RFID	RFID backscatter range	0.1m at 2m range with LoS	Not required	Medium RFID tag/reader price	Medium (highly dependent on backscatter signal strength)
UWB	Several tens of metres (75 metres baseline)	<2m at 75m range	Not required	High UWB anchor price	High (Strong against multipath, signal can penetrate objects, low power consumption)

Figure 2.2: Comparison between available positioning systems.

[9] RSSI's Role in RFID: https://blog.atlasrfidstore.com/rssi-role-rfid, 29 Dec 2019
[10] It should be noted that some materials offer better penetration for this radio frequency than others [22].

2.4.4 Future Developments

We also note that 5G networks may provide an efficient indoors positioning system in the future. It is documented to be able to achieve error margins of 2 to 3 cm respectively for Line-of-Sight and Obstructed Line-of-Sight through the use of beamforming technologies up to several metres [23]. However, it is important to note that as a growing technology, it has not yet matured. This leads to prices being high and a lot of devices being needed to cover the full breadth of the desired area of a warehouse.

2.4.5 Synthesis

Figure 2.2 summarizes the various positioning systems described above, revealing that the most interesting positioning systems would be the GPS for outdoor settings whereas either UWB or RFID RSSI would be the system of choice for indoor settings.

2.5 Constraints

As drones become increasingly available commercially and with enhanced performance in terms of payload capacity and range, we will undoubtedly expect an increase in the constraints regarding their use. We now address these constraints, as recently, regulations have been put in place to control their use in public and private spaces, along with indoors and outdoors-specific rules. Still, and regardless of current regulations, drones should be minimally safe to operate around humans. Besides the weight constraints seen above, propeller guards should also be accounted for to avoid serious injuries, specially when dealing with larger drones with more powerful propellers.

2.5.1 Physical limitations

Most notably, battery-life (and hence range) and payload capacity are the key features that limit a drone's operation and use. [24]. A practical test run described in [25] measured the cost of vertical movement such as the drone's lift against horizontal traversal finding the earlier to be substantially most costly. When contextualized for the specific purpose of warehouse inventorying, drones should limit their vertical lift when strictly necessary and rely on horizontal movement to traverse each shelf.

When using specific technologies for warehouse inventorying, drones can also rely on or benefit greatly from line-of-sight. The case for cluttered indoor spaces is usually a worst-case scenario. As discussed previously, Wi-Fi solutions will have their range and reliability hampered by the sheer amount of obstacles in a warehouse. Line-of-sight is also a major issue in some positioning and navigation methods, thus becoming an important factor to consider when developing a system architecture.

It is clear that current solutions always need to account for the physical scale of a project. Warehouses are typically larger than the average Access Point's range. Similarly, RFID solutions have to deal with the signal's penetration of objects reducing their effective range. Some positioning technologies like GPS can only be used in an outdoors setting, and since most warehouses are interior spaces, this has prompted the development of alternative solutions. These alternatives act on a local scale, come with their own infrastructure needs and do not exhibit the elegance of the easy to use and widespread GPS solution for (outdoors) global positioning. Both shelves and the items stored therein are not standardized, which adds a layer of complexity to navigation and sensor components of the drones.

2.5.2 Legal issues

Legal issues are a growing concern regarding the increasing ubiquity of drones. National aviation authorities tend to regulate the operation of drones within their borders[11] and the level of scrutiny can vary significantly between countries. The main concerns being public safety and the use of controlled commercial airspace. For example, in the United States users must obtain a Certificate of Authorization to operate a drone in national airspace.[12] According to the Federal Aviation Administration, drones must also be controlled by a user with direct line-of-sight to spot possible dangers. Indoor airspaces, however, are not public and therefore not regulated by the Federal Aviation Administration. It is unclear whether any regulation is in place concerning indoor airspace. Similarly, the United Kingdom's Civil Aviation Authority[13] determined that a drone or UAV operating within a closed-off airspace without possibility of "escaping" is of no danger to other aircraft operating in national airspace and therefore is not subject to any regulation.

Common sense dictates that small enclosed spaces are potentially more dangerous to humans. Yet, there seems to not be any regulations in place. Federal Aviation Administration and Civil Aviation Authority do not consider interiors to be navigable airspace, thus paving the way for a commercial application of drones in warehouses. It raises however concerns over the responsibility for accidents related to the use of drones indoors (in particular as typically these are workspaces) and to the possible future regulations yet to be applied over drones and their dimensions. Until a general level of regulation is accepted worldwide, drones for warehouse use will only be limited in their role in outdoors situations. In addition, if the drone is to be equipped with a camera, common workplace surveillance practices should be enforced. These also vary from country to country. Generally, employees should be aware of the drone's filming capabilities and it should be treated as a closed-circuit television system.

[11] Master List of Drone Laws: https://uavcoach.com/drone-laws/, 02 Jan 2020
[12] Federal Aviation Administration – Register Your Drone:
https://www.faa.gov/uas/getting_started/register_drone/, 09 Dec 2019
[13] Civil Aviation Authority – Recreational drone flights:
https://www.caa.co.uk/Consumers/Unmanned-aircraft/Recreational-drones/Recreational-drone-flights/, 09 Dec 2019

2.6 Related Work

The works considered to be the most relevant found in the literature are detailed below, both on the academic and commercially available level.

2.6.1 RFIDRead and drone-mounted inventorying

Bae, Sung Moon, et al, on "Development of Inventory Checking System Based on UAV and RFID in Open Storage Yard" [16] focus on inventorying and identifying misplaced items in an open-air storage yard. RFID tags carrying information like item count and item location placed on the stock are scanned by a remote-controlled drone carrying a portable Personal Digital Assistant with RFID reading functionalities. The Personal Digital Assistant collects data and the developed software can read tags, save tag information in a file, or clear tag data. Afterwards, the collected data is transferred and compared with the storage yard's database, with results matching four status outcomes: normal item, misplaced item, missing item and unregistered item. The system is capable of reducing errors and inconsistencies in the storage yard's inventory checking server by providing accurate readings with reduced labour and equipment costs. However, the drone still requires someone to control it. The storage yard is not roofed, therefore the drone can be easily located with GPS. This added challenge would need to be tackled if the system were to work indoors.

2.6.2 Indoor UWB localization for drones

Macoir, Nicola, et al, on "UWB Localization with Battery-Powered Wireless Backbone for Drone-Based Inventory Management" [4] showcase the feasibility of accurate indoors localization of autonomous inventorying drones utilizing Ultra Wide Band (UWB) anchor nodes focused on lowering deployment costs. The authors propose a new UWB Medium Access Control protocol that leads to large energy savings. The protocol itself uses Time-Division Multiple Access as the channel access method. This access method divides time into repeating superframes, each one containing slots which in turn relate to an action to be taken. The devices in range of the drone must be synchronized. It initiates the synchronization process with a sub-Ghz beacon message. All the anchor nodes receiving the beacon (containing the anchor nodes to be ranged) know when to activate their UWB radio, or otherwise sleep. After synchronizing, a polling message is broadcast to nearby anchor nodes. Then, each UWB radio is sequentially ranged. After the triangulation is finished, the report is forwarded from the drone to a Real-time Localization System with another sub-Ghz message, which calculates the drone's position.

The localization system reports a 5cm accuracy error margin. This requires errors in time-of-arrival in the sub-nanosecond range. For this, a three way message exchange (using Symmetric Double-Sided Two-Way Ranging [26]) is done to each anchor node. A single polling message is used to reduce the

number of messages per anchor node from 3n to 1+2n, wherein ”n” refers to the number of anchors. The system has been tested in a warehouse environment mock-up and an indoor running track with success. Yet, the UWB solution requires each anchor to be powered and an existing communication infrastructure. A next step to be considered would be the autonomous movement of the drone, given the level of accuracy of the localization method.

2.6.3 RFly and drone RFID relays

Ma, Yunfei, Nicholas Selby, and Fadel Adib, in their work entitled ”Drone relays for battery-free networks” [11], propose a system that seamlessly integrates with existing RFID infrastructures to bolster the effective range of its full communication capabilities to over 50m and a localization algorithm with an accuracy error averaging 19cm.

To do so, the drone carries a custom Printed Circuit Board to act as the relay between a centralized RFID reader and the passive tags. The relay is bidirectionally full-duplex to support backscatter communication and allows for the required four wireless transmissions at the same time: the RFID query (received by the drone relay), the relayed RFID query (received by the RFID tags), the RFID tag response (received by the drone relay) and the RFID tag relayed response (received by the RFID reader). The relay also preserves the timing of the relayed transmissions to ensure the localization system functions accordingly. The relay’s amplification of the received signal would otherwise lead to the same signal being picked up by the antenna on the drone relay itself. This self-interference is solved by having guard-bands generated to separate the reader and the tags’ frequency responses in the frequency domain.

To localize the drone, the challenges of phase entanglement and multipath are then addressed. The phase entanglement is solved by embedding an RFID tag into the relay itself, which serves a purpose of normalization to eliminate the impact of the half-link between the RFID reader and the relay. The drone is located by applying standard array equations since the drone’s movement emulates an antenna array. ”Antenna arrays exploit small phase changes due to distance in order to localize an object of interest”. A heatmap of the drone’s possible positions can be obtained particularly through non-linear projections. Accounting for multipath (represented by long streaks instead of a small but tall peak in the heatmap), the highest peak corresponding to the drone’s position is chosen. This work details extensively the requirements of operating an inventory system completely dependent of RFID and achieves its goal of creating a method of inventorying stock reliably and locating the drone itself with a high degree of accuracy.

2.6.4 PINC AIR

PINC AIR's warehouse drones for real-time inventory tracking by air [27] are an interesting commercially available example of an implementation of drones for inventory management. The available information on the technical specifications of the system is limited, but the drone is advertised to operate autonomously, avoiding obstacles in its path and using "optical, RFID and barcode sensor capabilities, to significantly improve the operational effectiveness and efficiency of warehouse inventory cycle count". According to the given information, the drone is capable of automatically performing inventory checks. It can determine the location of an item if needed and pinpoint its location. This system is versatile, but little information can be gleamed from its behaviour.

2.6.5 Synthesis

The focus of these works lies on the attempt to streamline warehouse item counting and routine inventorying, as well as developing methods for accurate localization of a drone in a challenging indoor environment (Figure 2.3). RFID for warewhouse sensorization has been used to great effect in the literature. Both UWB or RFID-based solutions seem viable for indoor drone positioning.

Work	System	Warehouse Sensors	Data Acquisition	Drone Positioning	Type
RFIDReader	RFID-based Inventorying system, drone-mounted reader, database	RFID reader	Collected on drone arrival	None	Academic
UWB anchors	UWB radio-based positioning system, drone-mounted polling device	None	None	Distance to synchronized UWB radios determined by message time-of-arrival	Academic
RFly	RFID-based Inventorying system, drone-mounted RFID relay to increase effective reader and tag	RFID reader, RFID relay	Relayed wirelessly to RFID reader	Antenna array equations exploiting changes in phase due to distance through non-linear	Academic
PINC AIR	Autonomous drone for warehouse inventorying	Optical, RFID reader, barcode reader	Unclear	Autonomous navigation, otherwise unclear	Commercial

Figure 2.3: Comparison between related works in terms of relevancy to the report.

2.7 Summary

We conclude the section on State of the Art by presenting the most relevant points made on the topics of drones for intralogistics, warehouse sensors, warehouse database updates, drone navigation,

constraints and related work.

Multicopter drones are viable candidates for both order-picking and inventorying tasks, the latter case more so, due to their small form-factor, their capacity to hover and their ease of operation.

Currently, barcodes and RFID tags (particularly passive tags) coexist in the warehouse environment, but generally RFID tags are more versatile due to not requiring line-of-sight, albeit being slightly more expensive. Active RFID tags boast much greater ranges but require a battery to be powered and are comparatively much more expensive. Optical sensors in current times are accurate but require specific lighting conditions and line-of-sight to the product identifier.

Drones can be expected to communicate with the warehouse's database by either using existing Wi-Fi infrastructure to send data in near real-time or by acting as a relay to a centralized RFID reader that broadcasts a query, greatly extending its range. Naturally, a drone can also be picked up and manually connected to a computer to transfer its data, it this is inadvisable due to human intervention and the partial loss of near real-time updates.

The main positioning system evaluated for exclusive outdoors-use was GPS, typically used in multicopters. However, as most warehouses are indoors other solutions arise. RSSI can be used to measure the drone's position through the signal strength received from nearby anchors. These can be Wi-Fi Access Points, Bluetooth devices, or RFID tags. The cheaper passive RFID tags allow for a much greater quantity of anchors to be placed for the drone to get a reading from, which in turn (and after filtering) allows for much lower error margins in measuring the drone's position when compared to Wi-Fi and Bluetooth devices. A UWB anchor system is also a strong contender for an indoors positioning system, as it boasts small ranging errors at much greater distances and does not require line-of-sight, all of this at a low energy cost. The system is also robust against multipath issues. When compared to the other solutions, it is expensive. In the future, it is expected for 5G to be able to produce more robust, accurate and efficient indoor positioning results.

There are several physical limitations to account for when considering the viability of drones for intralogistics and inventory management. The equipment mounted on a drone can severely hinder its already short flight time. It is also important to consider the cost of lift versus horizontal traversal, the latter being less energy-consuming. Drones should be kept at a safe distance of the shelves and should not pose any danger to the personnel working on-site, making propeller guards a necessity. Whichever positioning system is chosen must account for the unstandardized nature of different warehouses. In legal terms, drones are free to operate in enclosed spaces, yet the personnel should be warned if a drone has filming capabilities due to privacy protection. The personnel should also have training to handle the drone and to avoid potential accidents.

The proposition described in this report takes into account previous academic studies and current commercial ventures that have tackled the challenge of warehouse inventory management with drones.

3

Architecture

Contents

Architecture

A solution for the drone-based inventory management challenge can begin to take shape. This section will, given the assumptions below (section 3.1) and the analysis of the system's requirements (3.2), present an overview of the system and its functionalities (section 3.3). The system is subdivided into subsystems in section 3.4. Afterwards each system component is further explored and the requirements for each part of the drone inventory management system (section 3.5) is detailed. A brief summary of the design will be presented in section 3.6.

3.1 Assumptions

Listed below are a series of assumptions made about the environment in which the inventory management system will operate:

- **Drone Characteristics:** The drone is a remote-controlled multicopter capable of sustaining vertical and horizontal movement and carrying the required computational platform for the system. It is charged before being used to maximize operational time. It is only turned on when about to be launched, to stay at a low temperature and preserve battery. The drone is prepared according to the safety and security measures mentioned in Section 2.5.1 and 2.5.2. After completing its mission, the drone should safely return to its starting position.

- **Warehouse Physical Infrastructure:** The warehouse will be an indoor facility, with a configurable but well defined orthogonal layout for the drone to navigate. The warehouse will be set up according to plausible organization methods, which includes a dedicated Forward Area and Reserve Area, the latter of which the drone will operate in. The items can be stored individually or in a storage unit such as a pallet or box.

- **Warehouse Telecommunications Infrastructure:** The warehouse will have a database that the inventory management system can access and update and a control center from where it is operated. The warehouse is outfitted with a Wi-Fi network and is employing specialized workers

to manage the control center. If there is no networking inside the warehouse, the information is stored in a local database file on the computational platform.

- **Item Identification:** The items stored within this area of the warehouse must have a reusable digital identification method in order to reduce inventorying costs. The item identification method contains information regarding each item's position in the warehouse. This positioning information must be defined according to the warehouse positioning system that will be developed. The item's location is determined by the item position information written to the tag once the item is stored.
- **Warehouse Navigation:** The warehouse's corridors should be kept unimpeded and its shelves organized in such a way that allows for a consistent placement of item identification tags.

3.2 Requirement Analysis

3.2.1 Logistics

The inventorying system's behaviour will be constrained by the environment it operates in. A warehouse Reserve Area typically consists of rack shelves separated by aisles wide enough for a forklift to drive in. Each shelf is stocked with items in each of its levels. A real world example is translated into a simplified diagram (Figure 3.1) to illustrate the operational area of the inventorying system in a standard work environment.

The warehouse can be interpreted as a series of corridors between shelves. The shelves themselves are all oriented towards the same direction and each shelf can have several levels. Each level can contain several items stocked along its face. Listed below are a series of requirements the inventorying system should fulfill relating to its hardware and software functionalities, and to the way they interact with the system's surroundings.

3.2.2 Functionalities

The computational platform needs to be able to meet usability requirements and perform several tasks in order to function as an inventorying system:

- The system must be able to scan item identification tags located throughout the warehouse Reserve Area. These can be placed on an item or item storage unit and may be in line-of-sight, or obstructed by either another item, or farther inwards on the shelf it was stored in;
- The item identification tags should be affordable in bulk and ideally reusable, and their placement on the item (or item storage unit) should be simple;

Figure 3.1: Translation of a warehouse Reserve Area into a simplified diagram. The inventorying system operates along the aisles formed by the rack shelf layout (S1 through S12).

- Through the previous requirements, the system should be able to keep the warehouse inventory updated regularly;
- The system should identify misplaced items during the inventorying process;
- The system should function without interrupting the other logistics operations taking place inside the warehouse's Reserve Area;
- The system should require the least amount of human intervention possible to be handled.
- The system should be configurable for different Reserve Area layouts;
- The connection between the drone and the computational platform should have the minimum amount of downtime possible;
- The drone should unambiguously execute the orders it receives from the computational platform;

- The system should be able to correct the drone's course in the event of an unexpected error in its trajectory or behaviour, relying not only on the information it is receiving from the drone but also from the world around it.

3.2.3 Generic

The software to be developed must be flexible and adaptable.

- The software should be developed with an emphasis on relatively short response times in its communications with the drone and while processing items, in the seconds range;
- The software should access and update a local database file as well as updating the warehouse database, in case of losing its connection to the warehouse network;
- The communication with the drone should rely on a communications protocol with a degree of loss-tolerance;
- The route should be calculated prior to launch to avoid delays while the drone is airborne;
- The software should be flexible and route an orthogonal path given a warehouse layout of any size, adjusting the movement of the drone as necessary.
- A user interface should be available to configure and launch the system.

3.3 Functional description

Once configured according to the warehouse's Reserve Area dimensions, the smart drone can be set on its starting position and launched through the application by an operator. The smart drone proceeds to hover and begins to autonomously move according to a pre-calculated route through the shelves. This route is optimized so that the smart drone can cover the Reserve Area shelves with the least amount of movement possible. The drone moves from node to node (a node representing one of a series of intermediate points along an aisle) in the horizontal axis, only moving up to the next shelf level once the entire previous shelf level of the Reserve Area has been traversed. The smart drone continues this process until all shelf levels have been traversed. During this period, the smart drone scans the shelves immediately next to it along its path for items. The item information it senses is stored locally. If a connection to the warehouse database endpoint is available, it will attempt to update the warehouse database through Wi-Fi. If the drone happens to stray from its original course, it corrects itself either dynamically or once it reaches either a checkpoint in its path. After it finishes moving to the last node on the highest shelf level, it returns to its starting position by flying above the maximum shelf level and

moving directly towards its starting position's horizontal coordinates. After reaching it, the drone lands. It can then be retrieved by the operator. The operator can use the application interface to access and manage the information in the local database, configure a perimeter for the drone to operate within, or change the drone's movement configuration, such as the warehouse's dimensions multiplier or the drone's air speed.

The system overview in Figure 3.2 identifies the interactions between the drone carrying the computational platform and the warehouse according to the stipulations made previously.

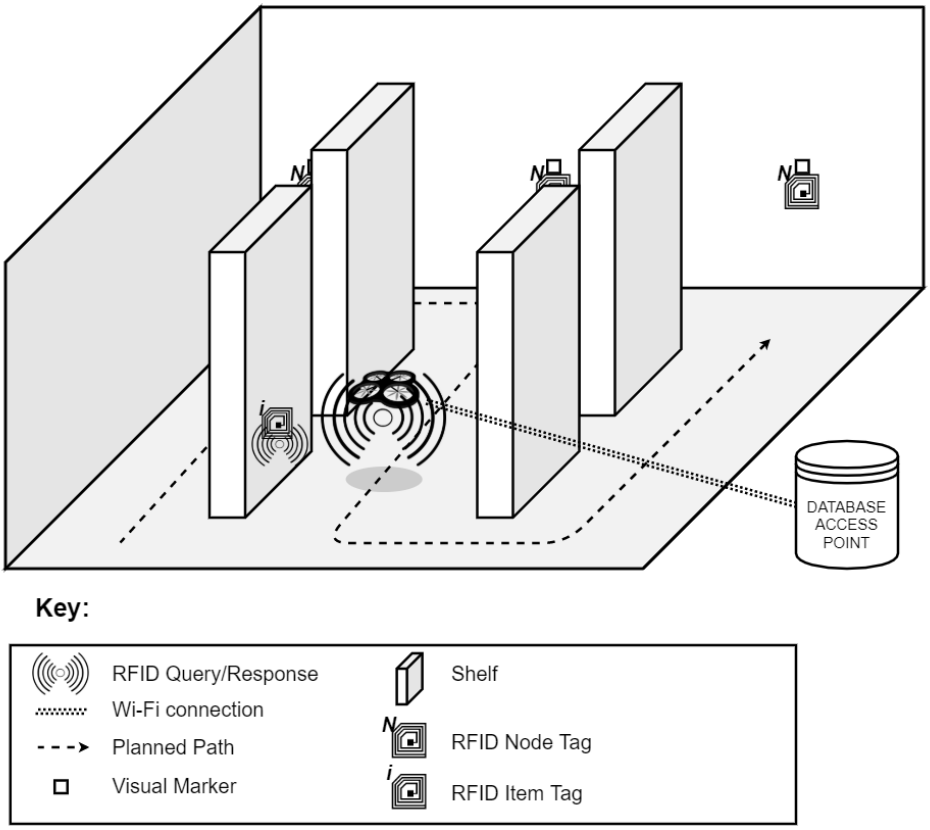

Figure 3.2: Three-dimensional system overview. The interactions between the computational platform and the warehouse sensor system and network are shown. This overview considers a checkpoint system to help the drone navigate the warehouse.

3.4 System Architecture

The proposed system can be schematized as a combination of subsystems: The smart drone concept encompasses both the airborne multicopter and the computational platform that provides it with the indoor navigation solution, the required inventorying functionalities and the transport and aggregation of necessary information between each subsystem. Hence, the application built on the computational

platform forms the backbone for the smart drone. It connects to the warehouse database and control center through the computational platform's hardware. The application itself manages its connection to the drone and to the warehouse endpoint, along with parsing the information received from the Warehouse Sensors and the Item Sensors. The basic system layout can be seen in Figure 3.3. Its sensing and navigation will utilize RFID technology (a reader and a set of programmable tags).

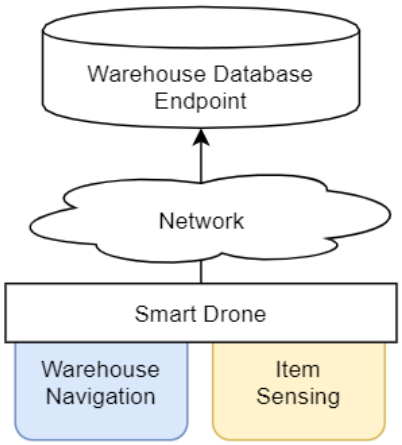

Figure 3.3: System layout. The smart drone performs item sensing and navigates the warehouse.

The computational system can be divided into four main subsystems. The warehouse sensor system, the drone communications system, the locally stored database and the navigation sensor system. The application will consist respectively of the inventorying component, the communications component and the navigation component. They are initialized by the application and access a centralized local database for shared information.

The components are pictured in Figure 3.4. The communications component serves as a bridge between the smart drone, its sensors and the warehouse database and control center. The inventorying component receives item scans and updates the local database accordingly. The navigation component calculates the route for the drone to take, and senses the environment around it to ensure the drone acting accordingly. The local storage is initialized by the communications component and stores two tables: a Node Table to be accessed by the navigation component, and an Item Table to be updated by the inventorying component.

3.5 System Components

Considering the complexity of the requirements, it is important to further specify the architecture of each subsystem and how it is connected as a whole. The inventory management system can be divided (according to the previous subsystems mentioned in the requirement analysis) into four main software

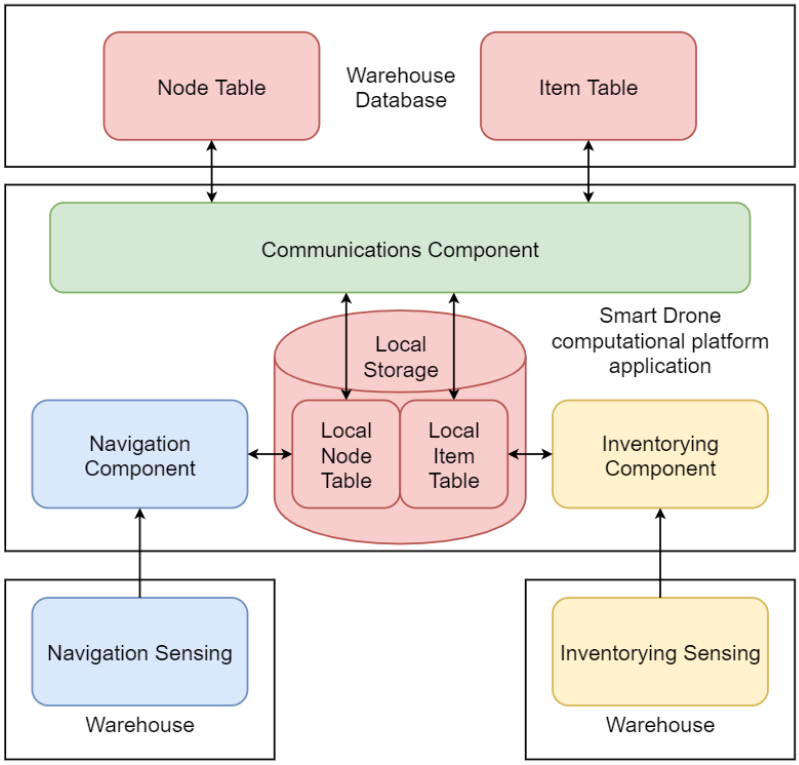

Figure 3.4: System component introduction: Both the navigation component and the inventorying component utilize the Smart Drone's sensors to gather information. The communications component relays this information to the warehouse database and control center.

components: the **inventorying component**, the **navigation component**, the **communications component** and the **database component**.

3.5.1 Inventorying Component

Firstly, the objective of the inventorying component (Figure 3.5) is to query the RFID tags placed on the storage units. The **Inventorying Algorithm** is a subroutine of the main logic operation focused on processing query responses from item tags. Once an RFID tag responds to the query, the item's Electronic Product Code (EPC), count and position are acquired. This information was previously stored in the tag's **Rewritable User Memory**. To identify misplaced items, the tag's expected position in the **Item Coordinates Map** is checked against its current coordinate set parsed from the tag's response. This will indicate if the item was misplaced or not. The local database is then updated with the retrieved information, along with the Flag attribute set to 1 to identify a misplacement. The local database (further described in Section 3.5.4) is a staging area the communications component accesses to update the warehouse database.

Several antennas may be required for the querying process, along with a robust protocol such as

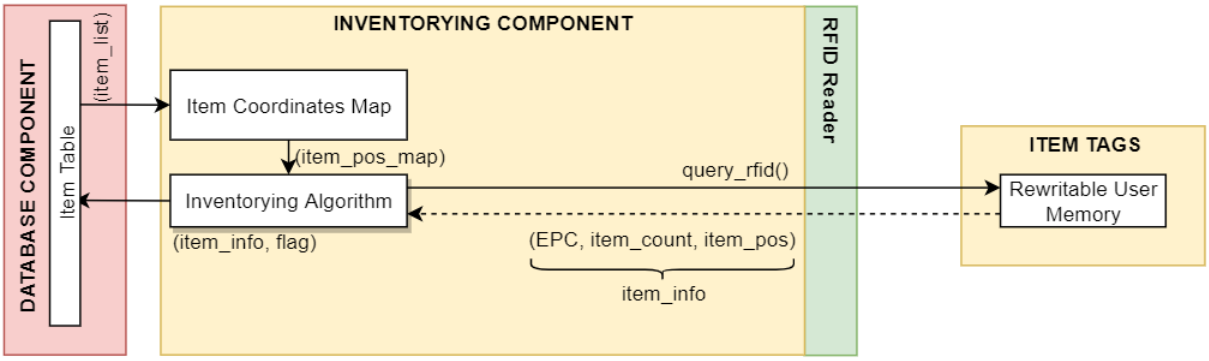

Figure 3.5: Software architecture of the inventorying component.

SBCS. The purpose of these protocols is to minimize the collisions between responses in the medium access. Collisions between the passive RFID tag backscatter signals require the reader to perform more query cycles, which spends substantially more energy and time, to receive every backscatter signal. Consequently, by applying this protocol the reader is less wasteful, quicker and more accurate when querying the tags in its vicinity.

Warehouse Sensor requirements: The warehouse sensors to be used are the passive RFID tags and a drone-mounted reader. The tags are inexpensive, reliable and widespread, making the solution versatile and easily set up in warehouses. The fact that line-of-sight is not required for this sensor technology makes it much more attractive than its alternatives, as warehouses are considered to be cluttered environments. Queries may also be performed whilst not immediately adjacent to an item, which further pushes RFID as the Warehouse Sensor subsystem of choice.

- **RF-ID sensor system –** Ideally, the RFID sensor system should be EPC Class 1 Gen 2 [28].
- **RF-ID tags –** The tags should be placed on surfaces that are not metallic, to reduce interference with the RFID signals. The tags chosen should have user memory programmed to allow for the desired configuration:
  - EPC;
  - Numeric indicator for the item's quantity in the storage unit;
  - Item Coordinates for the item's planned storage position.
- **RF-ID reader –** The RFID reader should be capable of both Signalling and Bulk Reading RFID query methods, in the 860 Mhz to 960 Mhz range (UHF). It is an established standard with a range upwards of 2 metres. It supports several readers and the Cycle Redundancy Check (CRC-5-EPC) error detection method is implemented[1].

[1] Specification for RFID Air Interface:
https://www.gs1.org/sites/default/files/docs/epc/uhfc1g2_1_2_0-standard-20080511.pdf, 02 Jan 2020

3.5.2 Navigation Component

The simplification of the layout of a warehouse (seen in Figure 3.1) is used to create a **Virtual Coordinates Map** to be used as a basic navigation method. By developing the Virtual Coordinates Map as a graph where the nodes are the intersections between aisles and the links are the aisles between shelves, we can progress towards developing a routing algorithm. The goal of such a routing algorithm should be to cover all the aisles with the least amount of movement possible. To do so, the warehouse's Reserve Area can be divided into its three basic measurements: its width (number of aisles, along an x-axis), its length (aisle depth, along a y-axis), and its height (number of shelf levels, along a z-axis). The routing process itself is an application of Dijkstra's algorithm for the shortest path [29] between the drone's current position and each of the map's unvisited nodes, keeping with the predetermined orientation of the shelves.The cost of each link between two nodes should be defined as the distance between them. The navigation component uses the Virtual Coordinates Map has a basis for its route. The **Navigation Algorithm** receives feedback from the physical world, which it uses to calculate its next movement (by being compared with the current route) or correct its current trajectory. This information is passed on to the **Movement Control**, which sends the commands to the drone.

The navigation component can be taken in two directions: the first one is dependent on bulk reading capabilities to obtain accurate RSSI reading from the surrounding tags in order to triangulate the drone's position. The second approach simply uses the RFID node tags as a confirmation mechanism for an otherwise blind navigation system reliant on the route determined by the navigation component. To complement the second option, the drone's built-in camera can be leveraged to perceive its surroundings and make adjustments to its course accordingly.

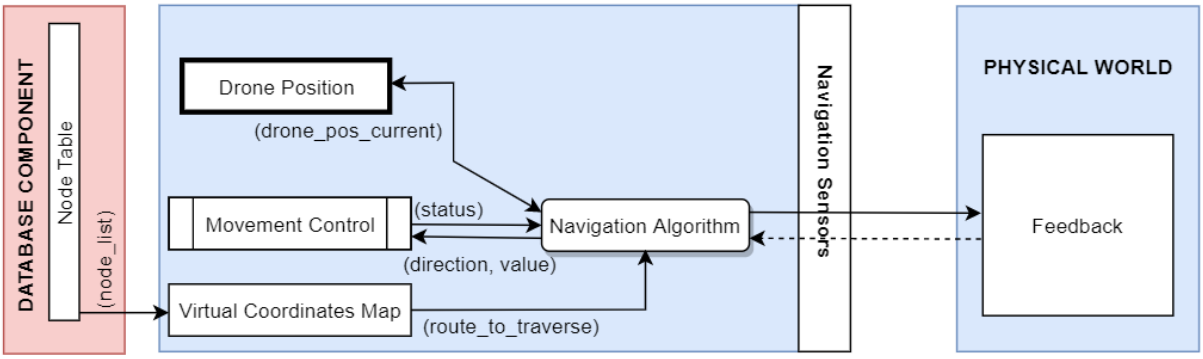

Figure 3.6: Software architecture of the navigation component.

RSSI Value Triangulation Approach: The navigation component (Figure 3.8) should take into account two different perspectives: the **Virtual Coordinates Map** and the **Path Marker and Path Boundary system**. As seen on Figure 3.8, the RFID reader should identify the path markers through signalling

and use the RSSI value from the nearest ones to compute its definitive position. The navigation algorithm triangulates the drone's position by way of the nearest path markers' RSSI values (polled according the Virtual Coordinates Map and the drone's current position), the drone's previous position and the distance moved during the previous movement of the drone. An example of the triangulation method is presented by Figure 3.7 with three path markers in use. Two main actions are performed:

- All three position indicators are combined to obtain a normalized drone position reading.
- The virtual coordinates map and the normalized drone position reading are used to compute a new route. The current route should be compared with the newly computed route. If a large enough deviation between them exists or if the drone is stationary for a long time (indicating an issue with the current route), the update to the route will be prompted.

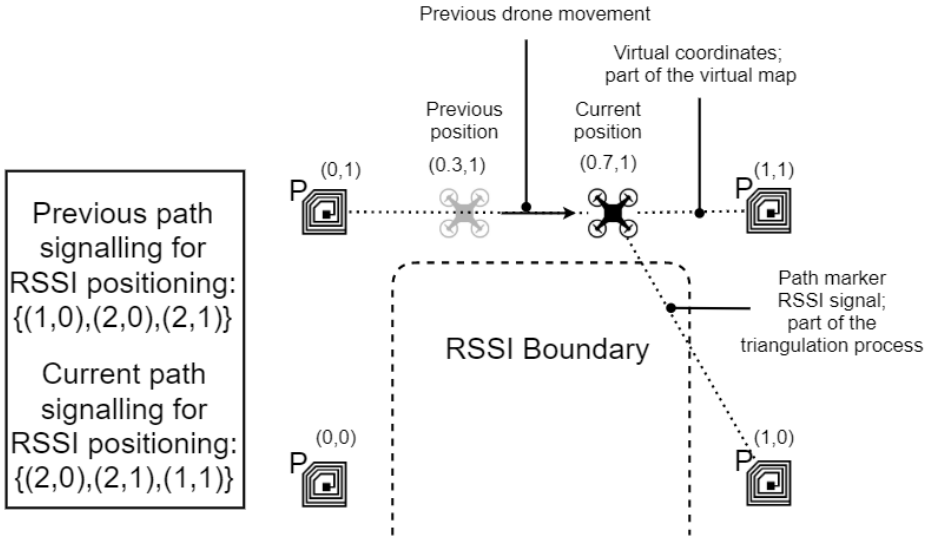

Figure 3.7: Top-down view of the RSSI Triangulation approach solution in an example configuration. Combination of the Virtual Coordinates Map (set of virtual coordinates), the drone's computed position by way of RSSI triangulation and the drone's previous movement.

Once the route is set, the drone should be instructed to move through the **Movement Control** according to the virtual coordinates of the next path marker and its current position. The drone's near constant speed should provide motion feedback to the navigation algorithm to be accounted for the normalization of the drone's position. The RSSI boundary set by the path boundaries should be respected when the drone moves. Its identification should be done by bulk reading the path boundary tags. The drone's movement control should stop it if several RSSI readings indicate that it is approaching a boundary.

RSSI Triangulation Requirements: For this approach, these tags should be divided into two sets to serve as **Path Markers** and **Path Boundaries**. The path markers should be programmed according

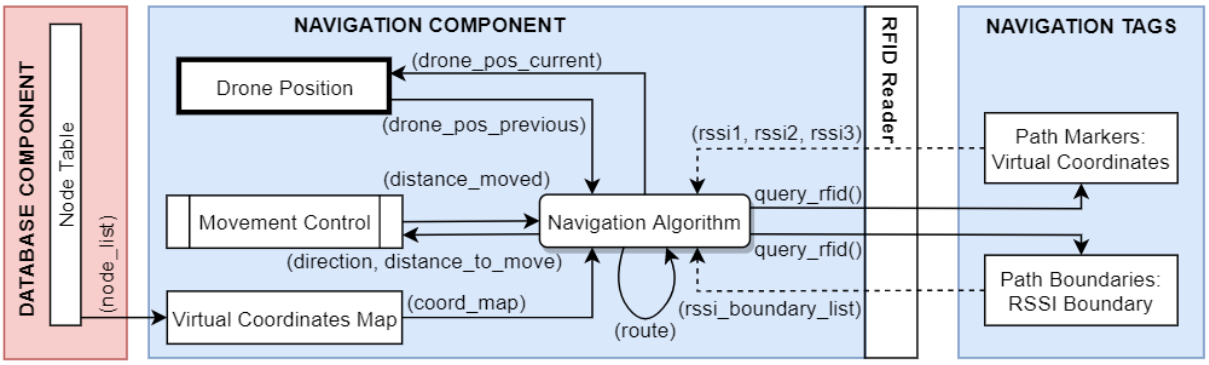

Figure 3.8: RSSI Triangulation approach to the software architecture of the navigation component.

with a node on the **Virtual Coordinates Map**. These will act as anchors for the drone to compute its distance to, through RSSI. The path boundaries can be placed onto surfaces where the drone should not go near, such as shelves. The drone should stop moving in the direction of the boundaries when a certain RSSI threshold is reached.

- The RFID Reader requires **bulk reading** functionalities;
- Similarly to the RFID requirements above, the RFID tags should be placed on non-metallic materials to avoid interference with the querying process;
- The tags should be placed without obstructions to maximize the RSSI accuracy;
- The tags should be placed with a consistent interval between them proportional to the size of each row of shelves, in order to increase redundancy and the accuracy of the drone positioning system.

Hybrid Checkpoint Approach: This option (Figure 3.10) involves a simpler process for the navigation component. By following a route pre-calculated by the **Virtual Coordinates Map**, the drone inventorying system does not require a constant and reliable connection to the node tags surrounding it. The drone moves through the aisle exclusively referring to the Virtual Coordinates Map for its **Movement Control** functions.

By using signaling instead of bulk reading, checkpoints have to be planned in advance to correct any eventual errors in the drone's trajectory. Once the drone reaches the end of an aisle, two subroutines begin. The first uses the **Camera Positioning System** to interpret the position of the drone compared to a **Visual Marker** placed on the wall at the end of the aisle as soon as it reaches the corresponding node. This subroutine indicates what movements should be made so that the visual marker is aligned with the center of the camera video output. The Movement Control corrects the drone's position accordingly. this can be seen in Figure 3.9. The corrections made to the drone's position are followed by the second

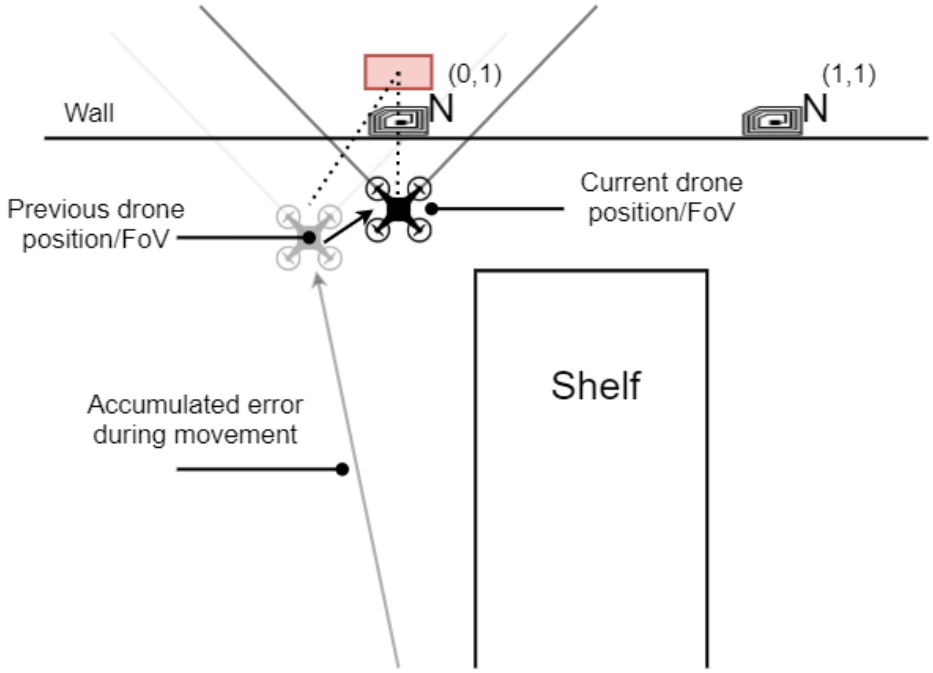

Figure 3.9: Top-down view of the Hybrid Checkpoint approach solution in an example configuration. Combination of the Virtual Coordinates Map (set of virtual coordinates) and the Camera Positioning System, complemented by a simple RFID position validation.

subroutine: a periodic check to the **Node Tag** buffer. If the expected coordinates are identified, the system considers itself to be ready to proceed with its movement to the next node.

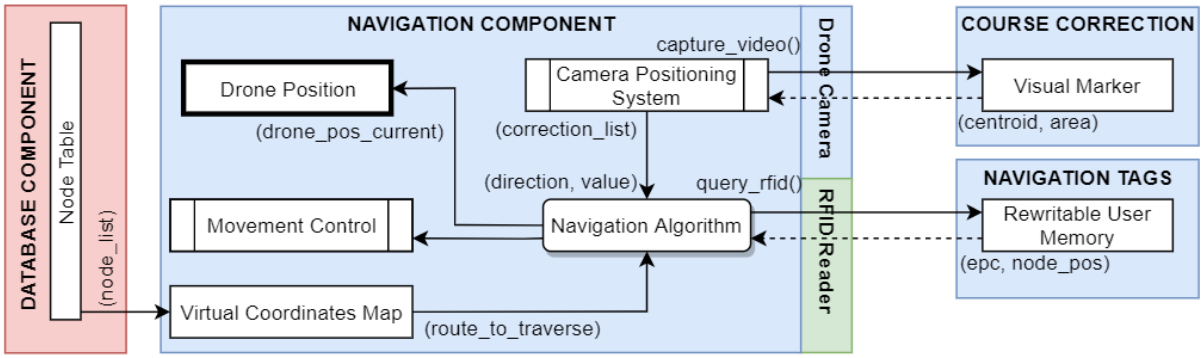

Figure 3.10: Hybrid Checkpoint approach to the software architecture of the navigation component.

Hybrid Checkpoint Requirements: For this approach, a single set of RFID tags will serve as node tags and complement a brightly coloured visual marker. The node tags should be placed on the wall at the ends of an aisle. The node tags each contain a set of two values for the horizontal coordinates where they are to be placed.

- The drone camera feed must be parsed correctly in order to obtain an accurate reading of the visual marker. This implies the use of existing computer vision libraries to process the incoming video transmission;
- As on the first approach, the RFID tags should be placed on non-metallic objects;
- Each tag should be placed near or on the respective visual marker.

3.5.3 Communications Component

The communications component's (Figure 3.11) task is to interact with the warehouse database and control center once started. The computational platform can be started and stopped in this component. While connected by Wi-Fi, it should update the warehouse database periodically with the local database and the drone's position through a reliable transport layer protocol such as Transmission Control Protocol (TCP). The main logic is listening for function calls by the control center. If it is stopped, the drone will cease its current routine and attempt to land.

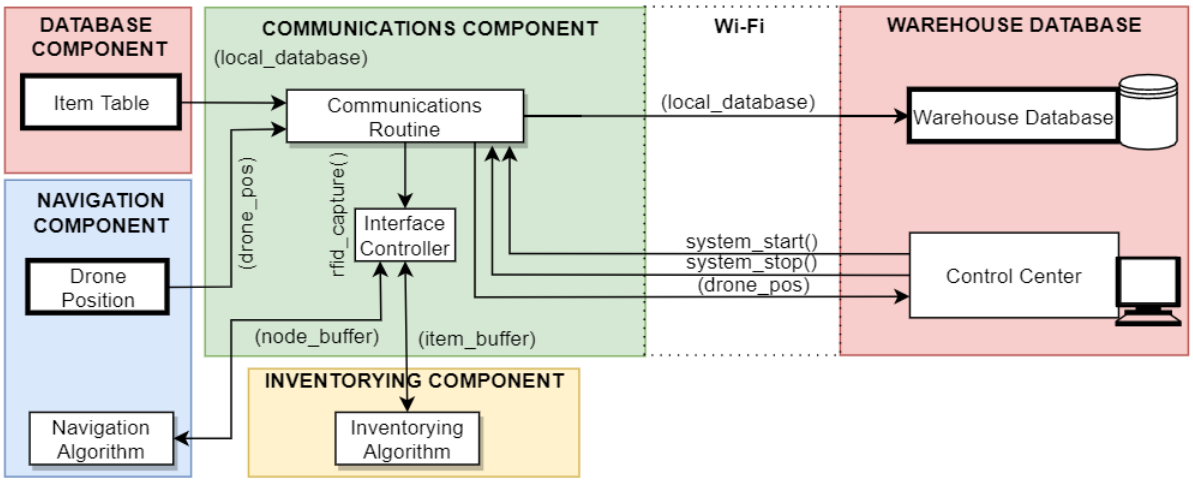

Figure 3.11: Software architecture of the communications component.

Warehouse Database Update requirements: The drone-mounted computational platform based on a single board computer, once connected to the RFID reader and the Wi-Fi network of the warehouse, should be capable of performing the following tasks of algorithmic nature:

- **Counting items** by having the RFID reader query its surroundings (part of the inventorying component);
- **Identifying misplaced items** by comparing the tag responses with drone's position (part of the inventorying component);

- **Updating the warehouse database** whenever a Wi-Fi connection is available (part of the communications component);
- **Knowing the drone's position** by evaluating its surroundings;
- **Traversing a warehouse** with varying layouts and dimensions;
- **Identifying path issues** during its automated inventorying, aborting the process if necessary (part of the navigation component);

3.5.4 Local Database

Finally, the database component, schematized in Figure 4.11, contains the information required for the drone to navigate the warehouse (Node Table) and a local database (Item Table) to keep the information retrieved from the scanned items. Item tag information is stored in the following format:

- **EPC**: The tag's unique identifier.
- **Item count**: The amount of items in the item container.
- **x-axis position, y-axis position, z-axis position**: The node tag position (x,y) closest to the item; the shelf level (z) it is on.
- **Flag**: The tag's misplacement flag. The tag is marked as a misplaced item if its flag value is 1.

Node tag information is stored in the following format:

- **EPC**: The tag's unique identifier.
- **x-axis position, y-axis position**: The node tag position (x,y).

The database-stored node tags are used to by the navigation component to automatically plot the drone's route, while the item tags will be stored throughout the drone's traversal of the warehouse.

Local Database requirements: The local database serves as a staging area for up-to-date item information to be sent to the warehouse database. The data should be persistent during the drone operation and easily cleared/overwritten.

3.6 Summary

The stipulated assumptions and requirements lead to a system architecture design based on a computational platform consisting of four software components. The inventorying component is tasked with

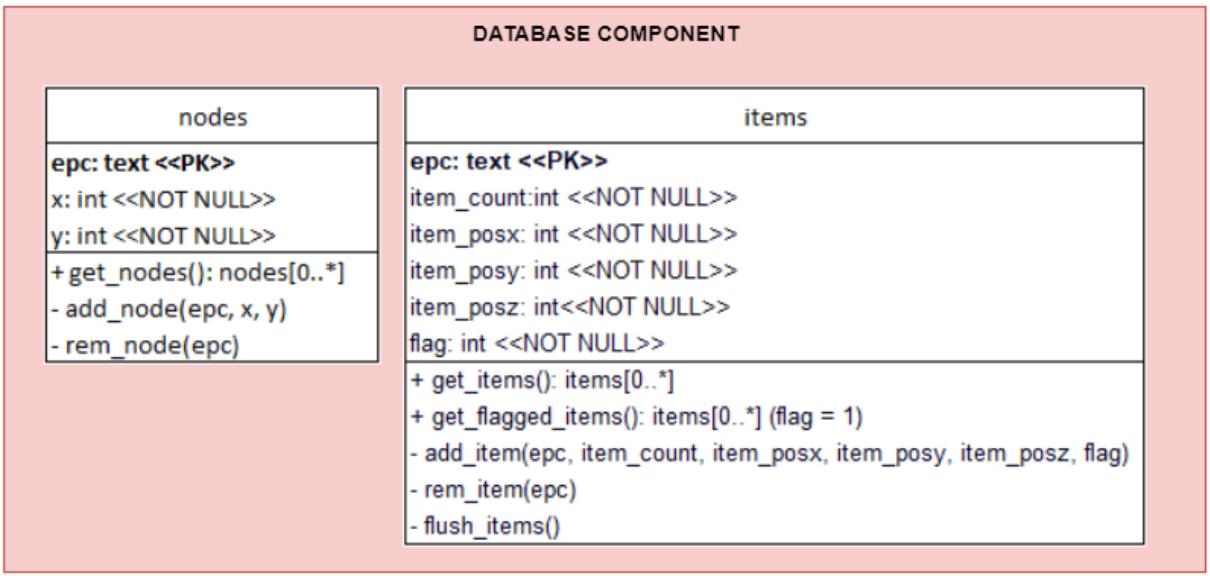

Figure 3.12: Local Database

querying the item tags to obtain the item data and flagging misplacements. The navigation component can be developed through two different approaches: in the **RSSI Triangulation** approach, the smart drone queries the path markers and path boundaries to validate the its movements and controls the traversal of the computed route; in the **Hybrid Checkpoint** approach, the drone camera is leveraged to, through the use of computer vision, provide adjustments to the drone's position at the end of each aisle it traverses. The communications component collects the inventorying component's Local Database to update the warehouse database and the navigation component's Drone Position to be reported to the control center. The locally stored database component keeps a record of the nodes to be traversed and an updated items table with near real-time information. The four components come together to form the software part of the computational platform. They are designed to operate on a single board computer.

4

Warehouse Inventorying System Implementation

Contents

4.1 General Design Options

The system architecture that was presented requires several hardware and software components that will be listed below. The drafted system architecture combined with the equipment selected for the task leads to a proposition for a proof-of-concept that is more constrained in scope than the original architecture. The outline of the system includes:

- **The Smart Drone**: composed of the proprietary Drone, the proprietary RFID Reader, and the Computational Platform (built upon an Single Board Computer (Single Board Computer (SBC))) connected to its subsequent components;
- **RFID tags**: identification tags for the items stored in the warehouse and the nodes which the drone uses to navigate it;
- **Visual markers**: a series of bright red $10cm$ cardboard squares to aid in navigation;
- **Network**: the parameters for the connections between the SBC and the drone, and between the SBC and the warehouse database access point;
- **Software architecture**: the logic and algorithms described in Chapter 3.

Each design option will be detailed throughout this chapter and lead to the finalized system assembly can be schematized as the following block diagram (Figure 4.1):

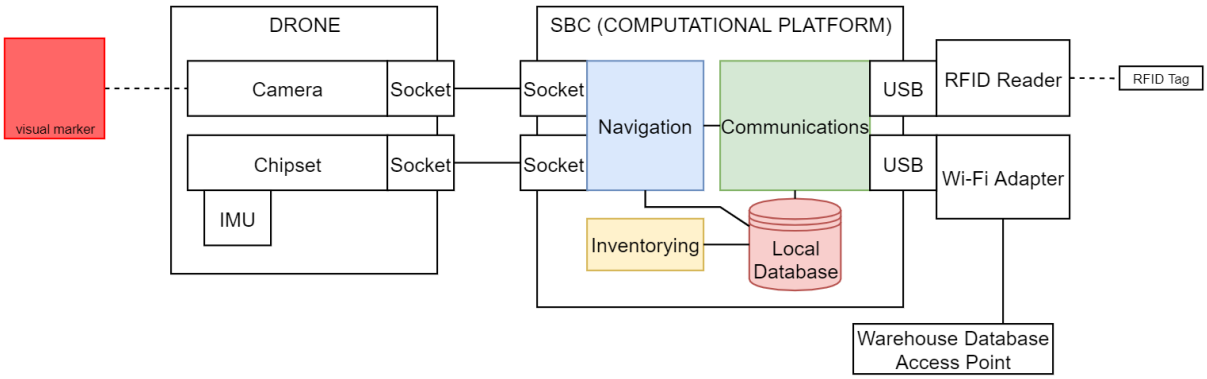

Figure 4.1: System Block Diagram

4.1.1 Smart Drone

4.1.1.A Drone

Ideally, the drone would be a multicopter, both programmable and capable of carrying a load of around $300g$ (accounting for the Single Board Computer and the RFID Reader). However, a compromise was needed since such a drone was not commercially available at the time.

The selected drone is a DJI Ryze Tello EDU, seen in Figure 4.2. It is a multicopter, programmable in Python, that weighs $80g$. The choice was based on the fact that the programming of the drone is far more important to demonstrate this work's proof-of-concept than its weight carrying capacity. It was selected with the prospect that drones will become more battery efficient and powerful in terms of how much weight they can carry, while retaining the ability to be programmed with their proprietary Software Development Kit (SDK). The information presented on this topic is found on Tello's specification documents, namely the User Manual[1], and was also derived from experimentation with the drone (further detailed in the Limitations section).

The drone is controlled by the SDK through a 2.4 GHz 802.11n Wi-Fi connection. Therefore, its connection to the computational platform will utilize the SBC's main network card. This implies the need for one additional network card in the form of a Universal Serial Bus (USB) Wi-Fi adapter to make the connection to the warehouse database. The drone does not have a GPS guiding system and is otherwise controlled through its Bluetooth connection to the TELLO proprietary Android Operating System application or Wi-Fi through the SDK. It is important to mention the drone is equipped with a front-facing 720p camera, which can be used as an additional method of navigation. The functionality of the Python 2 TELLO SDK will be detailed further in the System Design - Software section (Section 4.3).

Given a warehouse layout, the drone is expected to be able to move forward, backward, left and right, along with being able to gain and lose altitude and rotating horizontally (yawing) in order to perform turns once the end of the aisle is reached. The multicopter drone is expected to hover in place while waiting for further instructions. The drone is equipped with a status LED beside its camera that blinks green when executing commands, and yellow while it awaits instructions.

Figure 4.2: DJI Ryze Tello EDU

[1]User Manual: https://dl-cdn.ryzerobotics.com/downloads/Tello/20180404/Tello_User_Manual_V1.2_EN.pdf, 14 Dec 2020

Sensors: The drone is equipped with two types of sensor located on its underside. The first is a 3D infrared emitter/receiver pair. The second is a small camera which is part of the Vision Processing Unit[2]. Together, these sensors constitute the drone's Vision Positioning System (VPS). Whenever it is possible, the drone relies on these sensors to stabilize itself when airborne. However, this system is limited by several factors (further detailed in the Limitations section). In case of VPS failure, the drone enters Attitude Mode, which relies exclusively on its internal sensors (gyroscope, altimeter, barometer) to fly. In this mode, the drone cannot correct errors in its movement and the user manual advises an immediate and safe landing.

The drone's Inertial Measurement Unit (IMU) keeps track of its pitch, roll and yaw from the time it is turned on up to when it is turned off. It is possible to poll this sensor for updated IMU values. The IMU response is in integers which may not be sufficiently granular for precision movements.

A simplified schematic of the most relevant sensors of the drone is presented in Figure 4.3.

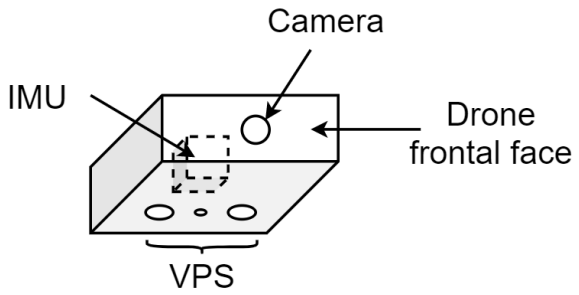

Figure 4.3: Drone Sensors. Below the drone, the VPS is composed of a 3d infrared emitter/receiver pair and the Vision Processing Unit camera. Inside the drone, the IMU keeps track of the drone's pitch, roll and yaw.

Limitations: When under use through the SDK (further detailed in Section 4.3, TELLO SDK), the drone has an experimental flight time of approximately 10 minutes (Figure 5.1) and reaches a maximum temperature of approximately $75^{o}C$ (Figure 5.3) when in use. The temperature of the drone is relevant since the drone does not have any cooling system besides the airflow provided by its spinning rotors once airborne. The drone counteracts its rising temperature by varying the speed on its rotors. This can cause the commands sent to the drone to be carried out with a larger error margin. If the drone's camera is in use, the drone must be airborne, otherwise it will overheat and stop responding to commands from the SDK. The drone enters a shutdown state once it reaches a critical temperature above $75^{o}C$ and stops executing commands. If it is on the ground, it will remain grounded and the status LED near the camera will blink red. If it is airborne, the drone will stop any movement it is executing, shut down its camera and Wi-Fi connection to the SDK and wait for the global safeguard timeout of 15 seconds to begin an emergency landing, in which it will descend vertically. During this process, the status LED

[2]Tello launch specifications: https://www.ryzerobotics.com/news/launch_news_page_en, 14 Dec 2020

near the camera will blink red. If the drone loses its Wi-Fi connection to the SDK on the computational platform, it will also enter the emergency shutdown state described above.

The drone takes approximately one hour to fully recharge its battery. Since its behaviour was noticeably more erratic when it was launched without a complete charge, the drone needed to be charged before every flight. This limited the amount of flights per day of work to less than ten. Its relatively short flight time and long charge time substantially increased the amount of time it took to develop and test each component, particularly the navigation component which was wholly reliant on the drone to be actively developed. While testing its main camera and video streaming functionalities, the drone was found to overheat and shut down after approximately one and a half minutes if it was not in flight. This setback also increased the time required to develop and test the navigation component.

The VPS keeping the drone stable cannot function properly under minimally disagreeable conditions. Of note (verified in an artificially lit room without any presence of wind and a homogeneous plywood floor, at day and at night) are the following:

- **Lighting conditions**: the drone struggled to stabilize itself (drifting left, right, forwards and backwards) when directly over a slightly darker zone than the average room illumination. Similarly, once hovering over a particularly reflective spot on the floor the same issue occurs.

- **Floor conditions**: the drone was set to hover over a neutrally-lit part of the floor. It kept a consistent altitude throughout 80% of tests. The height variation on the remaining tests was between $20cm$ and $1m$, which was significant. The problem regarding the drone drifting horizontally was also considered to be a problem relating to the floor conditions. In order to remedy this issue, a set of $70cm$ square plywood modules were placed on the ground. Their face was filled with black and bordered by two greens stripes. The drone was the most stable (keeping itself nearly motionless) when hovering near the border between the plywood modules and the floor. This indicates the VPS's preference for harsh contrast. It is not immediately clear whether the information from the infrared sensors takes precedence over the down-facing camera sensor.

- **Abrupt corrections**: If the VPS detects a change on the surface conditions below it, it will attempt to compensate by rising or falling. While attempting to retrieve the drone when it is wandering off course (due to the two points listed above), it is very hazardous trying to secure it before it crashes, as the drone suddenly changes altitude when it detects the operator's hand attempting to grab it. Once held, the motors rev harder as if to attempt to free itself. The propeller blades are sharp and this lead to minor injuries when handling the drone.

The drone does not compensate for its speed when stopping after a "move" command. Practically speaking, the drone will overshoot its movement target since it does not account for the time is takes to slow down from its current speed. This is noticeable even at the lowest possible speed ($10cm/s$).

The larger the distance the drone is commanded to cover, the larger the error becomes. Also of note is the drone's inability to move less than a predetermined distance, in this case $20cm$. The SDK does not provide a way to access the speed at which the drone is moving, only returning the value in centimeters per second it was previously set to move at. This value may not correspond with the current air speed of the drone as it could be speeding up or slowing down depending on if it has just started executing a movement command or if it is reaching its destination.

Furthermore, after some weeks of testing, the drone developed a problem with its rear-right motor. When taking off, the respective propeller was the last to get to full throttle. It was also the first to stop when landing. The remaining three propellers all started and stopped at the same time. When manually spun, the motor made a different sound to the other three motors. The propellers were replaced and the problem persisted. The motors were all clean and showed no signs of external wear. The most direct impact of this problem was a horizontal spinning motion that caused the drone to slowly rotate clockwise at a rate of approximately $0.21^{o}/s$ (determined in Chapter 5). This greatly reduced the drone's effective flight time during testing since the drone's unbalanced yawing caused very extreme inconsistencies between the drone's current heading and its intended course. Arguably because of this problem, the drone also had difficulties moving in a straight line, often drifting heavily towards its right (up to $1m$ right after moving $4.4m$ foward).

Related to this problem is the lack of IMU sensitivity. On startup, the drone resets its Inertial Measurement Unit. As mentioned above, the IMU records changes to roll, pitch and yaw in integers. Since the drone's yaw changes approximately $0.21^{o}/s$, this change does not appear to be large enough to notify the IMU of a change in the drone's attitude. To further elaborate on this point, the drone's spontaneous yawing is minute enough to not be recorded by the IMU, but large enough to be problematic after a long flight. This is an important observation, since IMU measurements of the drone's attitude in the order of a hundredth of a degree would allow for a near-perfect digital correction to the drone's physical problem if the IMU. The alternative is a much less elegant trial-and-error experiment to find a value near the spontaneous yaw to counteract whenever the drone is stopped for a considerable period of time. This large correction is enough to change the attitude values in the IMU, which implies all further calculations made when yawing the drone will have to take into account the accumulated offset gathered in the IMU until then.

To summarize, the drone functions best when at full charge, under consistent and neutral lighting, below its critical temperature and when flying over clear and contrasting floor markings.

SDK - Capabilities: The TELLO SDK was used as the basis for the system's navigation component. Its Python 2 implementation uses Python's native socket library to create two socket objects, both objects containing a (host, port) pair and utilizing User Datagram Protocol (UDP). The host value is the drone's

IP: ”192.168.10.1”; the port used for issuing commands is port number 8889, while the port used for receiving video feedback from the drone's camera is port number 11111. The use of UDP is clear when in terms of receiving a potentially lossy stream of video from the drone. However, the socket for issuing commands is also using UDP. This presumes that the application on the drone's side must have a failsafe for duplicate commands and commands received in the wrong order in addition to the validated existence of a drone-side acknowledgement response to issued commands. By default, the SDK aborts the command if the drone does not respond within the timeout window. The drone can respond to commands with an ”OK” message once it executes the command or with one of several error messages. The most notable error messages to be aware of are the following:

- **”Not Joystick” error**: The drone is still processing the previous command.
- **”Motor Stop” error**: The drone is not flying or, more commonly, exceeded the timeout for receiving commands, leading to an automated emergency landing.
- **”No valid IMU” error**: The drone attitude values are such that it cannot perform the command.

The SDK allows for a flexible range of commands[3] to be sent to the drone, the most useful of which are:

- **”command”**: Allows the drone to be controlled by SDK commands;
- **”takeoff” and ”land”**: Respectively launches and lands the drone into and from a stationary hover;
- **”streamon” and ”streamoff”**: Engages and disengages transmission of video stream from the drone camera;
- **”up x”, ”down x”, ”left x”, ”right x”, ”forward x”, ”back x”**: Henceforth collectively referred to as ”move” commands, they move the drone x centimeters in the specified direction. These functions are unfortunately limited to a minimum movement of $20cm$, making small precise movements difficult to accomplish.
- **”cw x”, ”ccw x”**: Respectively rotate the drone clockwise and counter-clockwise. Once again, the value of x is limited to a minimum of 1 degree, which hampers finer adjustments.
- **”speed x”**: Sets the drone's speed to x centimeters per second. This command's usefulness is also limited by its minimum acceptable value of $10cm/s$.
- **”battery?”**: Polls the drone's remaining battery percentage.

[3]Tello SDK 1.3.0.0 Documentation: https://terra-1-g.djicdn.com/2d4dce68897a46b19fc717f3576b7c6a/Tello%20%E7%BC%96%E7%A8%8B%E7%9B%B8%E5%85%B3/For%20Tello/Tello%20SDK%20Documentation%20EN_1.3_1122.pdf, 15 Dec 2020

- **”time?”**: Polls the drone’s time airborne for the current flight.
- **”temp?”**: Polls the drone’s chipset temperature.
- **”attitude?”**: Polls the drone’s pitch, roll and yaw.
- **”stop”**: This command is not listed but when transmitted, the drone will attempt to stop and return to a stationary hover.

Although there are ample commands to facilitate the drone’s movement through a orthogonal layout, the lack of granularity is the commands make precise movements challenging to execute. Otherwise, the drone executes the commands with relative accuracy, provided the VPS does not interfere with them and taking into account the extra braking time required for the drone, depending on its set speed and distance covered.

4.1.1.B RFID Reader

The RFID Reader is a WRD-130-U1 Ultra High Frequency RFID card reader (seen on the left side of Figure 4.10). It operates in the 860Mhz to 960Mhz range and is compatible with the EPC Class 1 Gen 2 type RFID Tags. Its keyboard emulation feature will be used as a means for the software to interpret the response once the RFID Reader queries a tag. The RFID Reader is also equipped with an proprietary SDK. The RFID Reader is itself heavier than the drone, but lighter options in the UHF range were not available. This implies having the drone carrying the RFID Reader will not be possible. Out of the box, the reader can read a tag placed in front of it, launching a visible cue (the default state red LED switching to a green LED for the duration of the tag’s permanence in front of the RFID Reader) and an audible cue (a single beep when the tag is first read). Once it receives the response from its query, the RFID Reader emulates keyboard operations for every byte it receives from the tag through its USB connection to the computer that is powering it.

Limitations: It was expected of the reader to also be able to read multiple tags at once through bulk reading. The SDK’s UHF RFID Reader Plus Protocol Application Example (Figure 4.4) implies the ability to perform bulk reading through the method seen on Figure 4.5. During testing, completing the three suggested steps before pressing the ”Auto read” button seemed to have no impact on the function. The expected values from each tag positioned in front of the reader (oriented horizontally in respect to the ground, facing the RFID Reader frontally, approximately 5cm apart) are not filed under the table’s ”NO”, ”EPC” and ”Count” columns. The SDK’s user manual is very vague in regards to the functionality of the bundled configuration application: when provided, the functionality descriptions are limited to labels for the buttons on the ”Setting” panel. It was not possible to extract any RSSI information from the RFID

Reader's queries on tags, neither through the provided configuration application nor through the use of Wireshark and the dedicated USBPcap plugin.

L<mask>,<action> <mask> lock mask 000~3FF <action> lock action 000~3FF	L<none or <OK>> or <error code> <none or <OK>> none: no tag in RF field <OK>: lock ok <error code> 0: other error 3: memory overrun 4: memory locked B: Insufficient power F: Non-specific error	lock memory
P<password> <password> access password 00000000~FFFFFFFF	P	set access password for R W L command, one time use
U	U<none or EPC> <none or EPC> none: no tag in RF field EPC: PC+EPC+CRC16	Multi-TAG read EPC

Figure 4.4: Excerpt from the SDK's UHF RFID Reader Plus Protocol Application Example. The first column indicates the command, the second column indicates the return message, and the third column indicates a brief description of the functionality. The last command pictured, "U", is of particular interest.

SDK - Configuration: The RFID Reader's configuration application was set up as follows:

- **"Setting" Panel** (Figure 4.6)
  - Unchecked "USB keyboard disable" option on "USB mode" tab;
  - Set "Key delay" to $1ms$ on "USB mode" tab;
  - Checked "Auto" option on "Read card mode" tab;
  - Set "Scan Time" to 1 x $10ms$ on "Read card mode" tab;
  - Set "Same Card" to 2 x $10ms$ on "Read card mode" tab;
  - Set "Power" to 18 on "Set UHF RFID Power" tab;
  - Set "Frequency" to "TW 922-928" on "Set Frequency Range" tab.
- **"Read" Panel** (Figure 4.7)
  - Checked "Auto read" option on "Auto read data" tab.

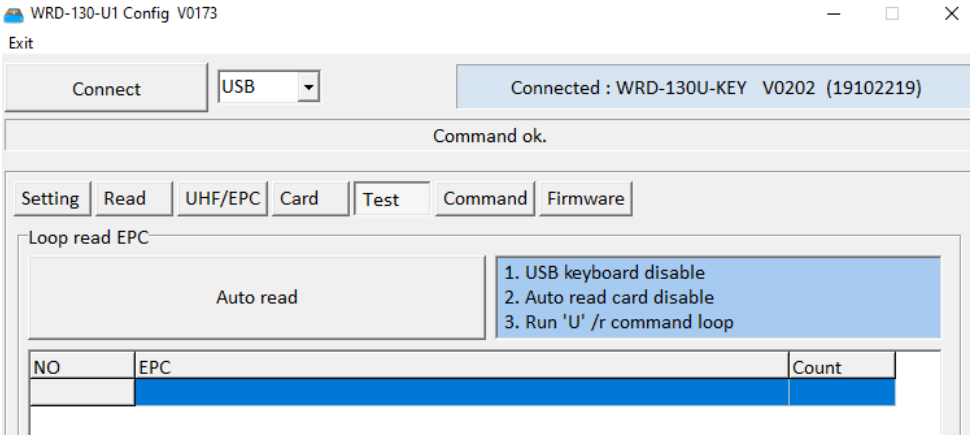

Figure 4.5: Test tab Auto Read Window to execute bulk-reading.

  - Set ”Read byte max” to 8 on ”Auto read data” tab.
  - Checked ”USER” option on ”UHF/EPC” tab.
  - Set ”End address” to 8 on ”UHF/EPC” tab.

- **”UHF/EPC” Panel** (Figure 4.8)
  - Checked ”USER” option on ”Read/Write EPC” tab.
  - Set ”Length(Word)” to 4 on ”Read/Write EPC” tab.

The tag readings will return 32 hexadecimal digits from the EPC field and 16 hexadecimal digits from the User Memory field. Every odd numbered packet parsed by Wireshark and the USBPcap plugin will contain one digit, resulting in a total of 96 packets per scanned tag.

SDK - Tag management: Using the RFID Reader’s configuration application, it is possible to write user information to tags through the following method:

1. On the UHF/EPC panel, on the text box to the right of the ”Write Tag” button, type in the information using Hexadecimal notation. If needed, the Length(Word) value can be increased or decreased, with each increment and decrement respectively adding or removing 4 digits from the text box.
2. After inputting the desired information, press the ”Write Tag” button.
3. Scan the desired tag with the USB-connected RFID Reader before the order’s timeout limit.
4. Once the ”Command ok.” status appears, the tag has been successfully written.

Similarly, one can read tags manually through the following method:

1. Press the ”Read Tag” button.

2. Scan the desired tag with the USB-connected RFID Reader before the order's timeout limit.

3. Once the "Command ok." status appears, the tag has been successfully read. The tag's information appears in the text box to the right of the button.

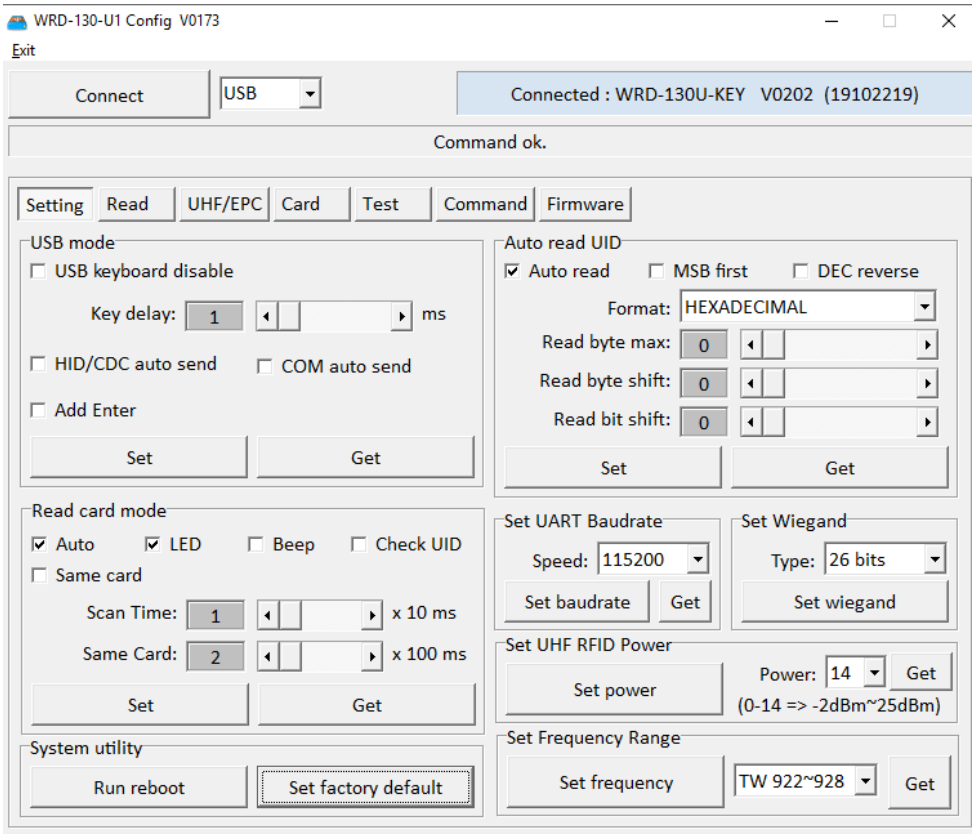

Figure 4.6: RFID Reader SDK configuration for the "Setting" Panel

4.1.1.C Computational Platform (Single Board Computer)

The Computational Platform is to be mounted on the drone in such a way as to not obstruct any of the drone's sensors, and ideally minimizing the displacement of the multicopter's original center of mass.

Raspberry Pi Zero The computational platform's software housing suggestion is a Raspberry Pi Zero. It has a small form-factor (65mm $\times$ 30mm $\times$ 5mm). It has a CPU clock speed of 1GHz and 512MB of RAM. It has wireless connection capabilities compatible with the 2.4GHz 802.11n wireless LAN protocol and can be powered by one of its mini-USB ports or by the following Raspberry Pi Battery Hat. The SD card that is used by the Raspberry Pi has a memory size of 16GB and runs Raspbian. The amount of memory available is important due to the fact that the computational platform will host a local database of variable size to store its inventorying results before being able to send it to the warehouse database.

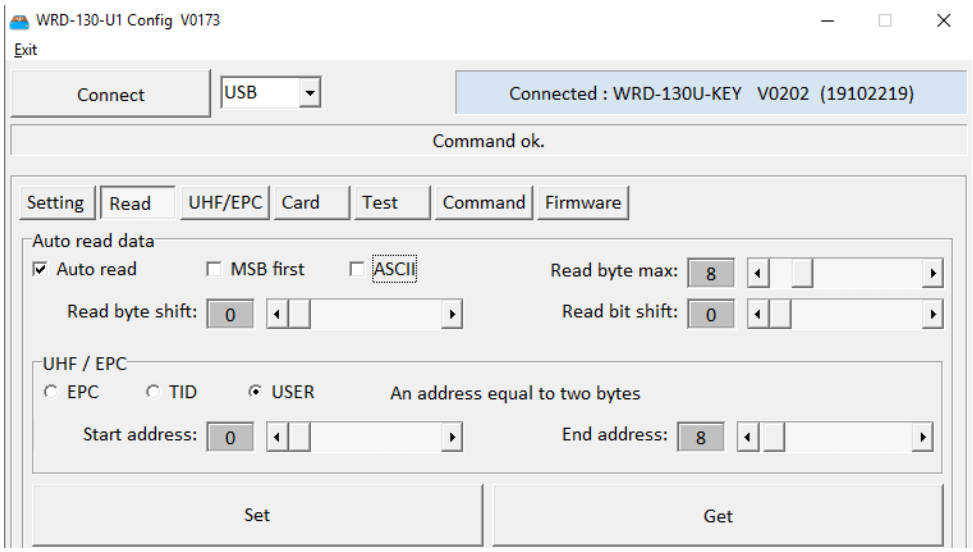

Figure 4.7: RFID Reader SDK configuration for the "Read" Panel

The Raspberry Pi can be equipped with a Hat that holds a Lithium Battery and provides power to the computational platform by being connected to the Raspberry Pi Zero through its 40-pin GPIO header.

For the purpose of developing the project, the Raspberry Pi Zero was switched out for a more powerful laptop. Ideally, the system would be built upon the Raspberry Pi Zero, which would be attached to the drone. Since this last part was not possible (the drone was too weighed down and unstable), a more versatile and easy to work with platform was chosen over the SBC.

USB Wi-Fi Adapter A Wi-Fi adapter is required to connect the computational platform to one of several APs via Wi-Fi for the transmission of the computational platform's local database to the main warehouse database. The adapter used is a TP-Link TL-WN321G 54 Mb/s Wireless USB Adapter.

4.1.2 RFID Tags

The selected RFID tags are ALN-9662, Higgs-3, "Short" Inlay type tags[4] (seen on the right side of Figure 4.10). They are operated by readers in UHF and comply with the EPC Class 1 Gen 2 protocol. The tags reportedly have a 96 bit EPC memory and 800 bits of non-volatile memory. The tags rely on proper orientation to be read (facing the reader in a horizontal fashion) and should be placed according to SOAP. Although the tags were advertised as compatible with European Union frequency ranges ($865MHz$ to $868MHz$), maximum read range testing in Chapter 5 indicate they operate almost exclusively in the Chinese ($920MHz$ to $925MHz$) and Taiwanese ($922MHz$ to $928MHz$) frequency ranges, with some success using the United States frequency ($902MHz$ to $928MHz$) as well.

[4] ALN-9662 Higgs-3 RFID Tag Specification:
http://www.alientechnology.com/wp-content/uploads/Alien-Technology-Higgs-3-ALN-9662-Short.pdf, 17 Dec 2020

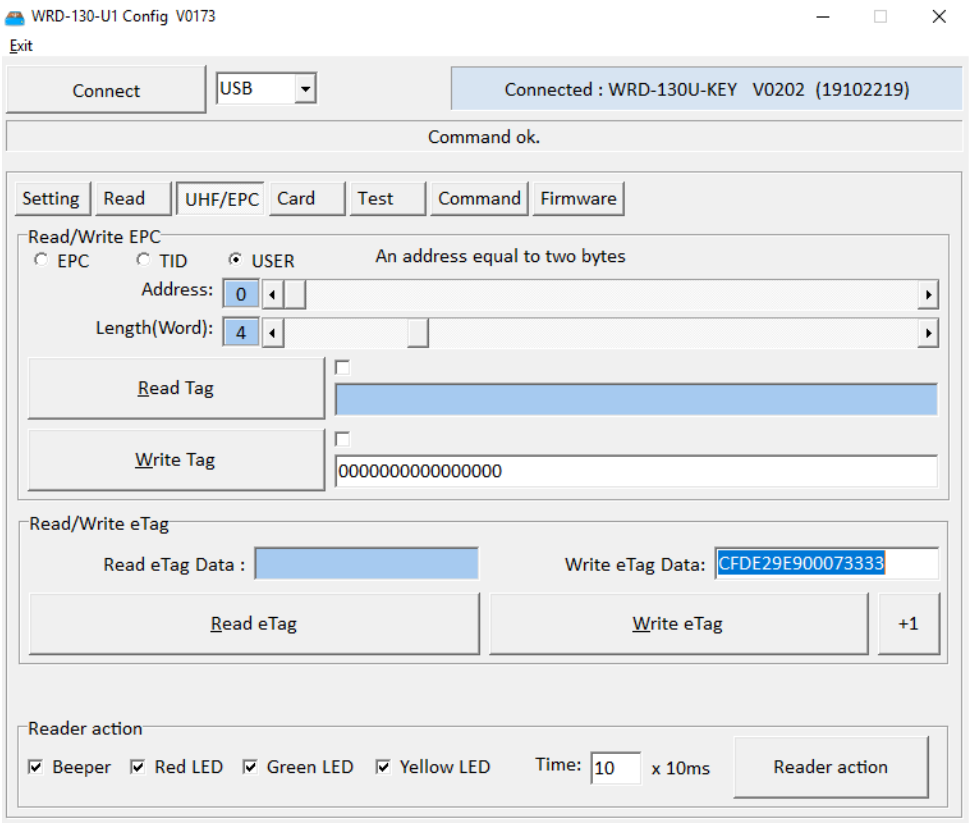

Figure 4.8: RFID Reader SDK configuration for the "UHF/EPC" Panel

Tag management: The hexadecimal information to be stored on the tags' user memory will depend on the tag type. The symbol distribution on the tags can be seen on Figure 4.9. C indicates item count (base 10); XS indicates the X-coordinates signal (positive or negative); X indicates the X-coordinate; YS indicates the Y-coordinates signal (positive or negative); Y indicates the Y-coordinate; Z indicates the Z-coordinate; F indicates the item misplacement flag and is either "0" (for correctly placed items) or "1" (for misplaced items), T indicates the tag type and is either "A" (for item tags) or "B" (for node tags). A . indicates unused information space.

Item Tag	C	C	C	C	XS	X	X	YS	Y	Y	Z	Z	Z	F	.	T
Node Tag	XS	X	X	YS	Y	Y	.	.	.	.	.	.	.	.	.	T

Figure 4.9: RFID tag User Memory information storage.

4.1.3 Summary

Given the hardware analysis, we can conclude that bulk reading will not be available for this project. This implies the use of the regular directional signalling method that is limited to one RFID tag at a time.

Figure 4.10: RFID Reader (left) and ALN-9662 RFID tags (right).

This impacts both the inventorying component as well as the navigation component.

Inventorying component The lack of bulk reading capability on the RFID Reader will cause the inventorying process to take longer and the placement of the item tags to be carefully planned. The item tags on a given shelf level should all be placed at the same height to maximize the effectiveness of the directional RFID query. The limited direction of the RFID Reader's queries also requires the system to be able to rotate the RFID Reader so that it can cover both the shelf to the left and the one to the right of the drone as it moves between the two shelves. A mechanically assisted swivel is assumed to be available for the proof-of-concept. The distance between each tag must be such that it ensures the system does not miss any items.

Navigation component Similarly, the lack of bulk reading reduces the drone's positioning system to one that relies heavily on single queries to individual tags. Since RSSI information is not available, the RSSI triangulation and boundary system approach detailed in section Chapter 3 cannot be developed. Instead, the Hybrid Checkpoint approach was chosen. The drone's built-in camera was leveraged as a means to gain more information on the drone's surroundings. The corrections that were to be made using the RSSI triangulation of the drone's position can still be made through the use of computer vision. The drone can adjust its position and trajectory at fixed location along its route. Namely, once the drone reaches the end of an aisle and is met with a wall, a **coloured square Visual Marker** on that wall can serve as an indicator to correct the drone's positioning. When the marker is captured by the computer vision module, it can be processed by the computational platform to send the drone a series of commands in order to move itself in such a way that is is positioned with its camera directly facing the center of the marker. The marker can be simply painted on the wall or be placed on it with an adhesive. To further ground the system in reality, an RFID tag can still be placed on the marker to supplement the

visual confirmation aspect of the Hybrid Checkpoint approach.

Given the weight of the required equipment and the maximum carrying capacity of the drone, the original setup will not be feasible. Instead, a laptop will serve as the base for the computational platform (replacing the Raspberry Pi Zero). The computational platform will be detached from the drone but keep the planned functionalities. The primary Wi-Fi connection will remain the one between the computational platform and the drone. The secondary Wi-Fi connection will be the one between the computational platform and the warehouse database.

4.2 System Design - Network

The drone requires a dedicated connection to its own Wi-Fi network to be controlled without manual use of the Android Operating System application. The selected SBC must therefore be able to connect to both the drone and a nearby access point for the warehouse database. The integrated network card in the single board computer will need to be supplemented by a second one, such as a network adapter connected via USB.

The commands are sent to the drone from the computational platform through UDP by design. This implies the connection is lossy and cannot guarantee the execution of every command sent. Ensuring no command packets are dropped is critical to operate the drone accurately. However, the drone sends an acknowledgement whenever it executes a command. This important feature is leveraged to build a re-transmission mechanism on the computational platform's side that ensures the command is re-sent if there is no received acknowledgement for it. If all planned retries for that command time out, a command is sent for the drone to land. Regardless, the drone engages an emergency landing procedure if it does not receive a command for 15 seconds. By having this re-transmission system last for 15 seconds, the computational platform's efforts to send the next command will last until the drone's safety feature makes it automatically land.

The local database is uploaded to the warehouse database via Wi-Fi as well. To guarantee the complete transmission of the item information kept in the local database, the transport layer protocol to be used is TCP. This connection was exemplified in the project with a mock-up File Transfer Protocol (FTP) connection.

4.3 System Design - Application Development

4.3.1 Python - Programming and interfacing

To limit the project's complexity, Python was chosen. Python 2 is already used for the drone's SDK. Since we require an interface for the RFID Reader, Python can also be used to interface with TShark (a network protocol analyzer). TShark (when coupled with USBPcap for use on the testing platform running Windows 10) can access the USB frame transmissions between the RFID Reader and the computational platform. Python's many publicly available libraries also make it ideal for prototyping system architectures connected at the software level such as this. However, while we can utilize Python 3 for most of the project, the drone's SDK relies on Python 2. Hence, any module that depends directly on the navigation component was developed for Python 2. The main consequence of this is the limited means to transfer information between Python 3 and Python 2 modules. While sqlite3 can be used by both Python version to query the local database, the tag.txt file is used as a buffer to store information between the Python 3 implementation of the communications component and the Python 2 implementation of the navigation component. The code base is available to be viewed and downloaded in a dedicated Github repository.[5]

4.3.1.A SQLite and Database Schema

Python's sqlite3 library allows for the creation of locally stored databases. It is used to store item tags and node tags, organized into their respective tables. The information stored in the tags is written to them by using the RFID Reader in "Write tag" mode and separates item tags from node tags in terms of content. This information can be retrieved through its keyboard emulation function. The queries to it are formulated in standard SQL. The output of a READ query can be stored in a list-type variable for further manipulation. The use of this library did not cause any problems with the rest of the implementation.

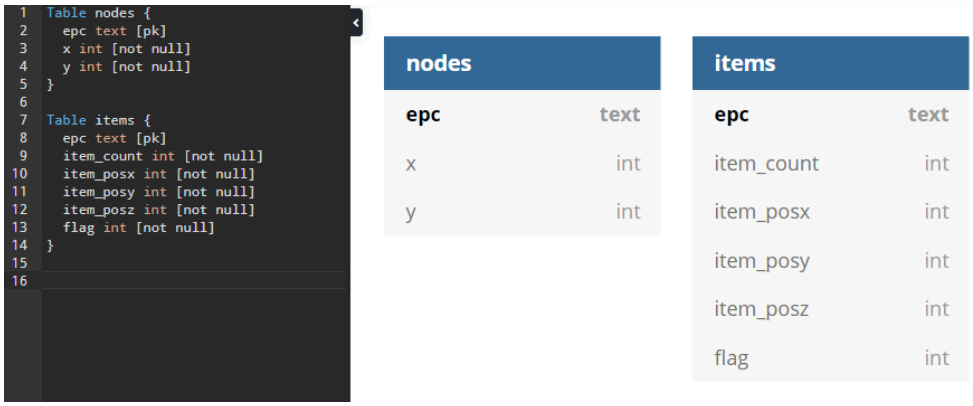

Figure 4.11: Database schema for tags.db

[5]Github Autonomous Inventorying System: https://github.com/D-K-O-I/thesis-autonomous-inventorying-system

4.3.1.B OpenCV

OpenCV is a popular open-source library that allows the system to use the drone's camera to detect the visual marker on a wall once it has reached the end of an aisle. Once connected to the camera, each frame received can be parsed to extract the marker's location in relation to the drone. By using the library, a dedicated logic loop was created to find the centroid of the marker and use that point of reference to issue commands to position the drone in relation to it. This is further detailed in the Computer Vision Hybrid Checkpoint Algorithm Implementation section.

4.3.1.C PyShark and TShark

TShark was the chosen network traffic analyzer for capturing USB traffic between the RFID Reader and the computational platform. Capturing frames and filtering them in real-time is not possible on the given USB interface, therefore a TShark instance will first run and store the captured frames into livebuffer.pcap. Afterwards, the Python 3 library pyshark is used to continuously read the USB frames sent by the RFID Reader which were captured in the buffer file in order to extract complete sets of USB packets. By having the keyboard emulation setting in the RFID Reader configuration application on, the filtered frames sent to the computational platform will each contain one symbol. The first 32 symbols represent the EPC, while the last 16 symbols are tag information to be processed. The last symbol received (part of the tag information) indicates whether a tag is a node tag or an item tag.

The library is not built to perform a live capture on the USB interface itself, and so must rely on running a new file reading every time there is an update in the capture file. This causes internal issues in the library which may cause some instability in the system.

4.3.1.D FTP Mock-up

The connection to the warehouse database is simplified into the ftp.py module, which connects to the localhost in order to upload the tags.db local database file. In a field test, this portion of the application would have been done via the Wi-Fi connection to the warehouse database access point.

4.3.2 Architecture Implementation

The system architecture described in Chapter 3 was implemented in three distinct Python modules (inventorying, navigation, communications components) and a database file (database component), which in turn may have other dependent modules. The communications component will be presented first as it serves as a base from which to launch or instantiate the other components.

4.3.2.A Communications Component

The communications component module (comp_com.py) is the initialization point of the inventorying system. Once started, the module initializes the tags.db database with two tables: ”nodes” and ”items”. Then, it launches a subprocess for a TShark instance to monitor and capture frames flowing through the USB connection between the RFID Reader and the computational platform. The output of this subprocess is saved onto the livebuffer.pcap file, which serves as a buffer to process the ordered frames from each tag.

Afterwards, the module's main purpose is starting the inventorying system cycle. From the start menu, one can also access the tag database to retrieve a list of all nodes, a list of all items, insert/remove items, insert/remove nodes, retrieve a list of all flagged items, and empty the item database (locally stored and temporary). When starting the inventorying system, the rfid_interface.py module is instanced and begins scanning the livebuffer.pcap file for tag responses. Until the drone lands, the communication component will wait for node tag responses. The navigation component must be launched separately. Once a node tag response arrives, its information is processed and sent to the (comp_com.py) module. There, it is written over the previous one on the buffer file tag.txt, which can be read by the separate Python 2 instance of the navigation component.

RFID Interface Buffer Implementation As the livebuffer.pcap file is filled with USB frames arriving from the RFID Reader USB port, the rfid_interface.py module utilizes the pyshark library to continuously read the buffer file. The frames are read one at a time in sequential order and passed through a filter. The display filter guarantees the received frames comply with the following rules:

- The frames contain 8 bytes of data length;
- The frames are sent from the selected USB port to the host system (the computational platform);
- The frame relative time to be evaluated is greater than 0.0 (avoids parsing the 24 initial USB frames relating to the device descriptor and configuration messages);
- The frames represent successful frame transfers (USBD status code equal to 0x00000000).

This filtering reduces the amount of frames to be processed per tag to 96 from the initial value of 192. Every frame contains a set of hexadecimal values that can be decoded with the translation table dictionary. This dictionary contains all the planned hexadecimal values as keys and the corresponding information as values. By parsing each frame sequentially, each new digit of information extracted is concatenated to a string for that tag. The last tag processed indicates whether a tag is an Item Tag or a Node Tag, according to Figure 4.9.

Information Flow to Navigation Component The comp_com.py module utilizes a queue object to share the tag variable with rfid_interface.py. The concatenated string storing the tag information is assigned to this variable whenever the last frame of a tag response is equal to ”B” (indicating a node tag response). Once it is retrieved, the main communications component module opens the tag.txt file in ”write-mode” and overwrites the stored information with the new tag information.

Information Flow to Inventorying Component Given its simplicity, the comp_inv.py module is initialized directly in the rfid_interface.py module. Whenever the last frame of a tag response is equal to ”A” (indicating an item tag response), the concatenated string variable storing the tag information is used as the input when the inventorying component's update_local_db function is called.

4.3.2.B Inventorying Component

The inventorying component is launched by the RFID interface module. It is not launched directly from the communications component module since the flow of item tag information is unidirectional toward the database. The inventorying component module comp_inv.py parses and evaluates the information of a given tag (checking the item's placement according to the information in the database) and inserts it into the database, updating the should the item already be present.

Database Item Table Automatic Insertion The inventorying component is automatically called by the rfid_interface.py module once it reads an item tag from the livebuffer.pcap file. The update_local_db function is called, with the item tag information string as an input. It is parsed according to the item tag information structure that can be seen in Figure 4.9. Afterwards, the extracted information is used as input for the insert_item function, which is similar to the one present in the comp_com.py module. The difference is the management of the flag variable: if the item already has a record within the database, its item position values (”item_posx”, ”item_posy”, ”item_posz”) are checked against the newly extracted values for that item. Inequalities will cause the sqlite3 UPDATE instruction to change the item's record for the ”flag” value to 1.

4.3.2.C Navigation Component

The navigation component module is to be started after the communications component module. On startup, a request is made to the tags.db database to retrieve all the registered nodes. This pool of nodes is passed as an input into a call for the vcm.py module, which returns a node map, a level-by-level horizontal route for the drone to follow. After the connection to the drone is initialized , the navigation component module converts each ”hop” from node to node into a movement command to be sent to the drone. Once the command is sent, the drone executes it and returns a response indicating the

command's status. When the drone reaches what would be the end of an aisle, two control flows are executed in order:

1. The drone is polled for its IMU attitude values. Of particular importance is the yaw value. If the yaw value does not correspond to the expected value for the drone's heading (taking into account previous corrections (the global OFFSET value) and its involuntary clockwise yawing previously described in Section 4.1.1.A), a correction to the drone's yaw is issued via a "rotate counterclockwise" command. The global OFFSET value is updated with this correction.

2. The OpenCV library is leveraged, which begins receiving the drone's camera and using its feedback to find the red visual marker on the wall ahead of it. This marker is used as a reference point for the drone. The algorithm will send movement instructions to the navigation component in order to have the marker as close to the center of the camera's capture as possible

3. Finally, it awaits a physical RFID tag confirmation from the visual marker, timing out if none is received.

This process is repeated throughout the course of the planned route. At the end of the route, the last few commands orientate the drone towards its origin point and attempt a landing once it has reached its destination.

The navigation component leverages threading for four tasks:

- Receiving responses from the drone at all times;
- Before adjusting the drone's position, to fix the drone's yaw;
- Before adjusting the drone's position, to start a 30 seconds timer for the Hybrid Checkpoint algorithm to operate;
- While adjusting the drone's position, to prevent auto-landing due to longer processing times.

Virtual Coordinates Map Algorithm Implementation The Virtual Coordinates Map is built from the existing nodes table in the database. The VCM class is instanced once during every execution of the system by the navigation component. All node records are then added to the VCM and represent each entry of the class's nodemap variable. Afterwards, the route_3d function can be called. This function creates a route that passes through all nodes in each shelf level (route function calls), starting from the bottom level. The routing algorithm is a modification to the DJP algorithm [30] with weighted vertices (nodes) and the idea of a "lazy" minimum spanning tree. In each bi-dimensional plane corresponding to each shelf level, the objective of the algorithm is to traverse all vertices with the least amount of effort (distance to traverse) possible and completely avoiding branching paths. To do so, a single-branch

minimum spanning tree is created for a given shelf level. The single-branch spanning tree can be built by keeping track of reached and unreached nodes, but only considering the distances of the unreached nodes to the last reached node (the node the drone would be on at that point in time). Since shelves are orientated in rows along the system's y-axis, this is used to attribute differing weights to nodes in different rows to that of the drone's current one.

When selecting the lowest effort hop, the algorithm considers the effort required to reach each of the drone's nearest orthogonal neighbours. When the lowest effort hop is executed, the now covered node is removed from the list of unreached nodes (which itself starts off equal to the list of keys in the nodemap with the starting point removed). As long as the nodes belonging to the aisle the drone is currently on are not completely covered, the lowest effort node to hop to will always be present on said aisle. Only once the edge of an aisle is reached does the algorithm consider hopping to the next aisle. The edge-case where the drone starts away from the edge of the aisle must be considered. This would lead to a situation where the drone would have to return to the aisle to cover the remaining nodes. The algorithm opts to reach the other edge of the aisle if it is still not covered, instead of switching aisles once at an edge. Hence, the weight given to y-axis movements should be unitary, while x-axis movement weights should be equal to the amount of nodes per aisle. If the drone finds itself in the center node of an aisle without prior traversal on that particular aisle, it will first traverse towards the highest y-value by default. Similarly, if there is an equal number of available aisles to the left and to the right of the drone once the current shelf is fully traversed, the drone chooses to head towards the lowest x-value by default.

Once a node is covered, it is removed from the list of unreached nodes and added to both the list of reached nodes and to the spanning tree. The current position is updated to the coordinates of said node. This process continues until the list of unreached nodes is empty. The route function returns the lazy minimum spanning tree, which is the ordered list of nodes that the drone is to traverse for the minimum effort coverage of that shelf level in the warehouse. The route_3d function adds the spanning tree resulting from each pass of the algorithm to the traverse_order variable. This variable is accessed by the comp_nav.py module to execute movement commands.

Computer Vision Hybrid Checkpoint Algorithm Implementation When the drone reaches the edge of an aisle, the Hybrid Checkpoint algorithm engages. Firstly, the drone adjusts its yaw in order to directly face the wall. The objective of the module is to identify a $10cm$ wide square red visual marker on the wall in front of the drone's camera and to provide the navigation component with movement controls using the marker as a reference point. A UDP connection to the drone's video streaming port is initialized. Each frame of the video capture is converted to a HSV version from which a colour mask is made by using a set of calibration sliders. The mask is then arithmetically multiplied by the original frame (bitwise "AND" operation) to obtain the frame_result variable.

In order to obtain a contour of the shape in frame_result, it is set to grayscale, then blurred and finally filtered through a binary threshold. From here, the OpenCV findContours[6] function automatically obtains the contour of the shape of the marker. The contour retrieval algorithm chosen was RETR_EXTERNAL which selects the extreme outer lining of the contour, and the contour approximation algorithm chosen was CHAIN_APPROX_SIMPLE, which compresses diagonal, vertical and horizontal contour segments into endpoints. This choice of approximation is important due to the fact that many frames will be processed in rapid succession and memory limitations should be taken into account.

Afterwards, a bounding rectangle is formed using the contour and a centroid for the shape is placed at the center of the bounding rectangle. Then, the vertical and horizontal components of the vector defined by the centroid of the shape to the center of the frame are calculated, along with the area of the contour. These variables are then passed into the identify_correction function call. The three types of corrections expected are:

- **Horizontal sideways movement corrections**: The horizontal value norm between the center of the captured screen and the centroid of the marker is greater than the established limit. This causes a ”move left” or ”move right” command to be issued.
- **Vertical movement corrections**: The vertical value norm between the center of the captured screen and the centroid of the marker is greater than the established limit. This causes a ”move up” or ”move down” command to be issued.
- **Horizontal forwards/backwards movement corrections**: The area of the bounding rectangle for the marker’s detected shape is too small or too large. If it is too large, then a ”move backwards” command must be issued, since the drone is too close to the wall the marker is on. If it is too small, then a ”move forwards” command must be issued, since the drone is too far away from said wall.

A fourth correction, based on the angle of approach of the drone to the marker was considered, but ultimately replaced by polling the drone’s attitude value for its yaw and deriving a yaw correction command from it. Regardless, a description of the original approach follows: Given the drone might not perform two 90^{o} turns perfectly when switching from one row of shelves to the next, it is important to verify if the drone reaches the end-of-aisle marker at an unexpected angle. If the drone reaches the marker within acceptable angle deviation values, the bounding rectangle surrounding the marker will be very close to a perfect square. The more the angle the drone is at over the acceptable limit, the more noticeable will the distortion of the shape of the marker will be, causing the bounding rectangle to be wider than it is tall. The issued commands are limited to ”rotate counterclockwise” and ”rotate clockwise” until the distortion is minimized. This has two problems however: the first is that it may require more

[6]OpenCV findContours function: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html, 15 Dec 2020

corrections of the remaining three types given the new position of the camera in relation to the marker; the second is that given the distance of the drone to the wall, the angle deviation might not be noticeable.

The corrections are issued for every frame that is captured (the module is currently set to capture a frame whenever it finishes issuing commands for the previous frame in order to minimize overload). The connection to the camera is closed once the drone begins its hop to the next node to reduce processing time. The processed video feedback (Figure 4.12) is available in the application, but is partially disabled in field tests, possibly due to how the library image processing functions. Regardless, the video feed is processed and the correct commands are issued to the drone.

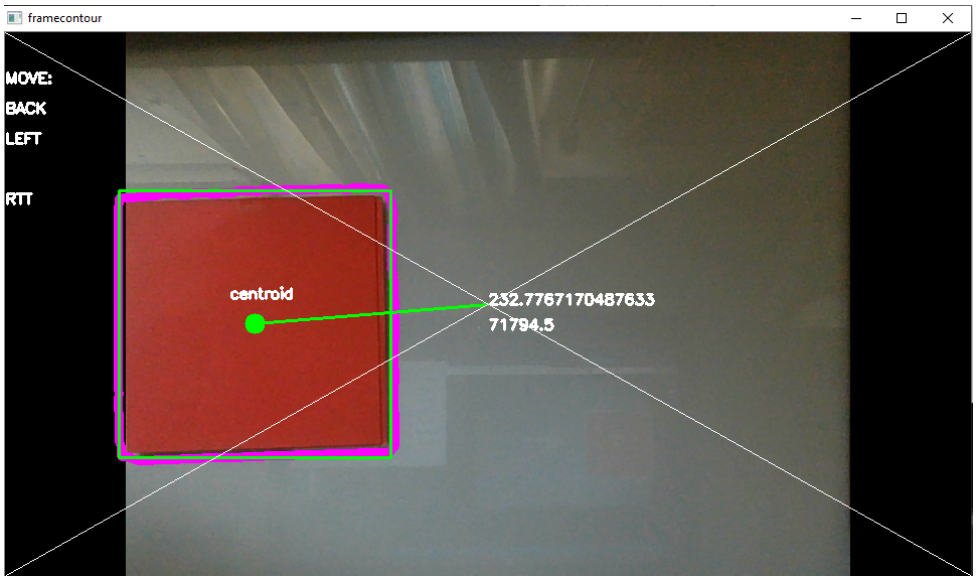

Figure 4.12: Video feedback processed through OpenCV. Featured in the top-left are the movement cues (”BACK”, ”LEFT”) for the centroid to reach the center of the video feedback. The vector norm from the center of the screen to the centroid is pictured in the middle of the screen. Below it is the computed area of the shape defined by the green bounding rectangle.

Drone Positioning Verification The tag.txt buffer file, accessed as ”read-only” by the navigation component’s main module, confirms the drone’s position after adjusting the drone’s position. This information flow originates in the rfid_interface.py module, where any node tag that responds to a query has its information written to the tag.txt buffer file, overwriting the previous tag information stored. There is a failsafe in place to ensure the drone continues its planned course after waiting for the 15 second timeout without reading the correct node tag from the tag.txt buffer.

All existing records of node tags are retrieved from tags.db in order to be used by vcm.py’s routing functionality, which returns a 3D-route for the navigation component to follow.

4.3.2.D Database Component

The database component was implemented using the sqlite3 Python 3 library, utilizing the database schema suggested in Chapter 3 and Figure 4.11. The database is stored in the tags.db file. The primary keys for both tables are a tag's EPC. These are unique to each tag and serve as a way to ensure no two tags are representing the same coordinate space.

Information Flow to and from Communications Component The database component can have the item and node tables edited or read by the communications component through SQL queries. The queries can perform insertions, selections or removal of items and nodes on their respective tables.

Information Flow to Navigation The navigation component launches a query to obtain a list of all nodes in the node table. The table is returned as a list of tuples (each representing a node, its EPC and its coordinates) for the navigation component to input into the vcm.py module.

4.3.3 Code Architecture

The application's communications component was developed according to its original design: it serves as a centralized module that is a user interface and can interact directly with the local database. However, both the inventorying component had to suffer some changes in order to accommodate the hardware limitations discovered in the drone and the RFID Reader.

The inventorying component kept its original functionality: parsing tag information to update the local database while identifying misplaced items. Its use is more limited due to the inability to perform bulk reading. It relies on having tags arranged in such a way that they complement the drone's movement (placed in a horizontal line).

The navigation component, although maintaining its original virtual coordinates map design for the routing portion of the warehouse traversal, saw changes to its positioning system. Instead of using bulk reading and comparing RSSI values from node tags to triangulate the drone's position, a simpler system relying of the drone's camera and RFID signalling was developed. Once the smart drone is facing the wall at the end of an aisle, its uses its camera and the OpenCV library to determine what corrections to the drone's position should be made such that it remains safely in the center of its trajectory. Besides horizontal and vertical movements, polling its attitude allows for correcting the drone's yaw. To further anchor this system to the physical world, the RFID Reader can be used to scan a node tag on the visual marker to verify the drone's position.

Information Flow Figure 4.13 details the flow of information inside the system, the type of information being circulated, and which module is using it as input.

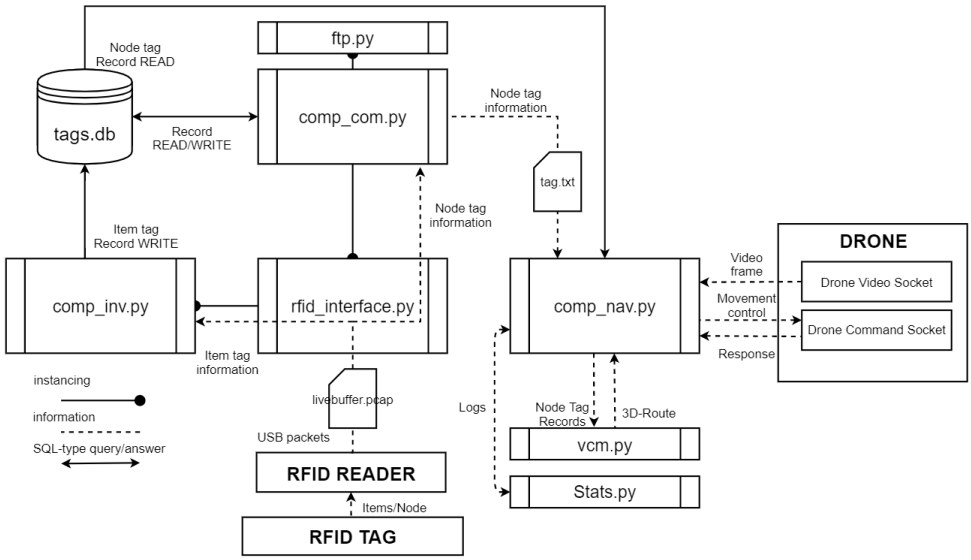

Figure 4.13: Software design - Python Implementation

Information enters the system by three ways:

- Manual writing of data (item tag information, node tag information) in comp_com.py which is inserted directly into the tags.db database;
- Automated retrieval of data (item tag information, node tag information) from livebuffer.pcap: if the data represents an item tag, it is used as input for the comp_inv.py module, and used to update the database's items table. If the data represents a node tag, it is sent via a queue object to comp_com.py which writes the tag's information as a string to the tag.txt buffer file to ensure compatibility with the Python 2 comp_nav.py module.
- The comp_nav.py module receives command responses from the drone via a Wi-Fi UDP static connection between the computational platform and the drone.

The system outputs information through the following means:

- comp_com.py, when data is read from the tags.db database by the user;
- the computational platform's USB port, first to query the USB ports available and once the RFID Reader begins scanning tags to respond to the transmission of each USB frame.
- comp_nav.py, every time a command is sent to the drone via the above-mentioned Wi-Fi UDP static connection.

The software developed supplements the shortcomings faced in the hardware section. Mitigating accumulated movement errors throughout the drone's traversal of the warehouse is possible with the

use of OpenCV, and therefore was the chosen approach. It is partially limited by the locations at which the drone can correct its position. Therefore, in order to avoid stacking compound errors throughout its autonomous movement the drone must stop at the end of each aisle and whenever a turn is required of it. While there might be some margin of error after each aisle is traversed, the position where the drone should be can be reached with relative ease.

4.4 Summary

The warehouse inventorying system will consist of five parts: the **Smart Drone**, the **RFID tags**, the **Visual Markers**, the **Network**, and the **Software Architecture**.

The smart drone consists of a multicopter that is programmable in Python, an RFID Reader and a Single Board Computer. The multicopter has a series of sensors which help it remain stable in flight (VPS, IMU) and a frontal camera, used to capture images and video. The drone's stability is heavily influenced by the VPS, which relies on specific lighting and ground surface conditions to be precise. The drone's limitations were described in detail, through observations and experimental results. The drone's SDK is described.

The RFID Reader's capabilities and limitations are described in detail. All changes to the RFID Reader's settings were made through its bundled SDK and documented. Preliminary tests determined its inability to use bulk reading on RFID tags. The SDK is used to manually write information to each RFID tag. Being limited to signalling each individual tag implies that the inventorying process will require the tags to be placed all at the same height and with an interval between them to minimize any loss of information. The lack of bulk reading forces the use of the Hybrid Checkpoint approach previously described in Chapter 3.

The SBC is ideally a Raspberry Pi Zero, equipped with a Wi-Fi adapter to provide the second Wi-Fi network connection the inventorying system requires. Since the SBC could not be attached to the drone without disrupting its movement, a more powerful laptop was opted for instead.

The RFID tags were divided into a set of item tags and a set of node tags. Each type of tag has relevant information written to it. Figure 4.9 illustrates the setup used.

The system network can be divided into two $2.4GHz$ 802.11n Wireless LAN networks: one is generated by the drone and ensures the connection between the drone and the SBC; the other establishes a link between the SBC and the Access Point for the warehouse database.

Application-wise, the software relies on the use of some existing libraries and applications. The navigation component is written in Python 2 to ensure compatibility with the drone endpoint. The remaining components are written in Python 3 due to their own dependencies. Python's sqlite3 library is used to interact with the local database. Regarding the Hybrid Checkpoint navigation solution, OpenCV is

required to parse and process the incoming video frames from the drone. Pyshark and TShark are used to parse the USB packets arriving from the RFID Reader. A simple mock-up for a warehouse database updater was written to transfer files via FTP.

The inventorying system requires two modules to be started: the comp_com.py module allows for interfacing with the database and starts the inventorying component; the comp_nav.py module is tasked with navigating the provided warehouse layout according to the Hybrid Checkpoint approach.

5

Testing and Results

Contents

Testing and Results

5.1 System Characterization

5.1.1 Unit Tests - Drone

The multicopter was tested with the goal of understanding the practical limitations of its performance. Three main tests were conducted to plot the evolution of the drone's remaining battery, its SNR to the computational platform and the drone's internal temperature. The drone was made to hover in place for 8 minutes while its battery percentage, signal-to-noise-ratio to the computational platform and internal temperature were measured. The drone was polled every ten seconds. Wi-Fi latency is the source of the cumulative delay observed in the response times. It is also important to note that the drone camera was on (this increases the rate at which the battery is drained). The details regarding each test follow.

5.1.1.A Battery Expenditure

Figure 5.1 shows a sampling of responses to battery queries transmitted to the drone. The vertical axis represents the remaining battery percentage (22 to 100). The horizontal axis represents time in seconds (8 to 488). Over the course of 8 minutes, the drone consumed 78% of its battery life. The remaining battery percentage falls almost linearly, which indicates the drone would most likely expend its remaining battery over the next two minutes (leading to an automated emergency landing). By comparing the experimental flight time with the flight time provided in the official specifications page[1], it is clear that the drone does not meet the desired expectations for its autonomy. However, this may be due to the camera being in use. It is important to have it turned on since a significant portion of the warehouse navigation will rely on it.

Flight Time There is an almost direct proportionality between the flight time and battery expenditure. The observed battery expenditure represents an approximate 26% increase over the expected battery expenditure given the theoretical flight time of 13 minutes at the 8-minute mark. If the information of

[1] Tello EDU technical specifications: https://www.ryzerobotics.com/tello-edu/specs, 18 Dec 2020

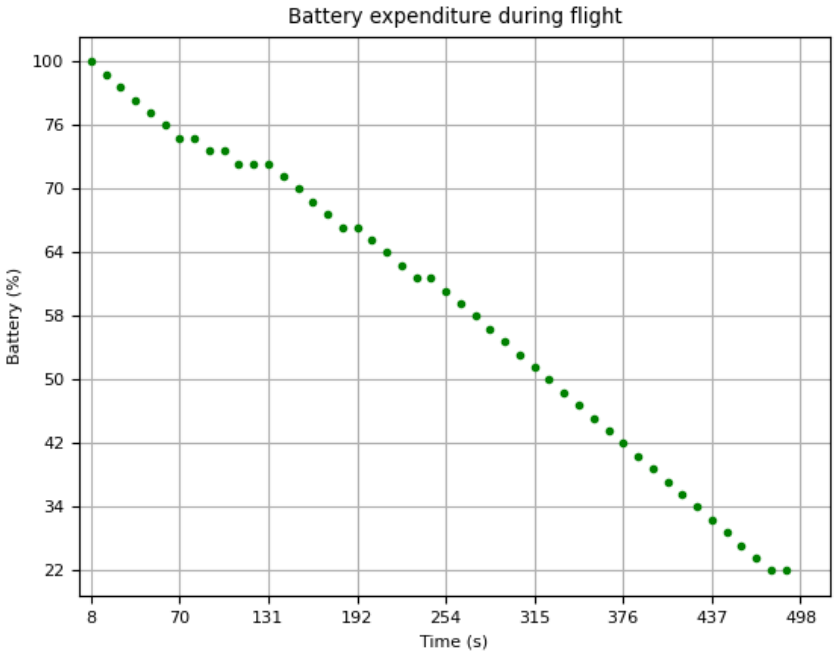

Figure 5.1: Drone battery expenditure on stationary hover mode.

the graph were to be extrapolated to a full battery discharge, the maximum flight time would be under 10 and a half minutes. Since the drone must return to its starting position, it is important to leave approximately 20% of the battery for the return trip or to re-position itself for an emergency landing. Therefore, the drone has an effective flight time of approximately 8 minutes. To verify the extrapolation above mentioned, the drone's maximum battery autonomy was tested. It was timed from the moment of takeoff until it landed. The drone remained airborne for approximately 10 minutes and 14 seconds. The significant difference between the theoretical flight time and the observed flight time may also be due to the wear on the drone battery.

5.1.1.B Signal-to-Noise Ratio

In Figure 5.2, the vertical axis represents the SNR which remains constant throughout the test. The horizontal axis represents time in seconds (8 to 488). This result is expected since neither the drone nor the computational platform change positions relative to each other except for during the initial takeoff. Considering the ideal situation where the drone carries the SBC, the SNR would remain constant.

Connection Quality The drone maintained the Wi-Fi connection to the computational platform while registering a constant SNR. This result was expected since the distance between the computational platform and the drone is constant.

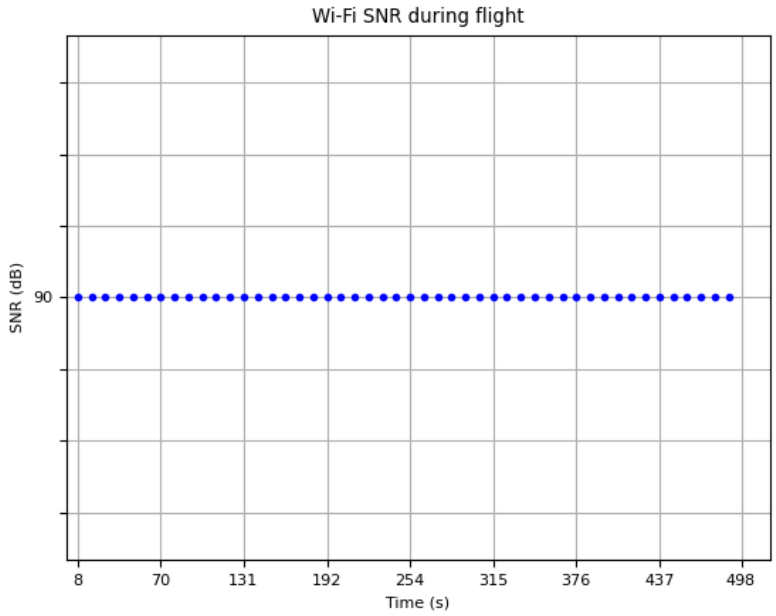

Figure 5.2: Drone to computational platform connection signal-to-noise-ratio on stationary hover mode.

5.1.1.C Internal Temperature

Figure 5.3 shows a fluctuating reading on the drone's internal temperature. The vertical axis represents the drone's chipset temperature in degrees Celsius (58 to 74), averaged from the 3^oC interval received in each response. The horizontal axis represents time in seconds (8 to 488).

Chipset Temperature The temperature measured on the chipset of the drone increases when it is under use, as expected. Once the temperature reaches between 72^oC and 74^oC, it stabilizes. The drone has no internal cooling mechanisms, so this is due to the quadcopter blades spinning faster in order to fan cool air onto the chassis (the sound produced by the drone varies when in flight, even when in a stationary hover, possibly caused by this cooling event). This is compounded by the fact that the drone overheats after spending approximately two minutes turned on without taking off.

5.1.2 Functional Tests - Drone

The drone was tested in a well-lit indoor environment over a patterned floor. When the first stationary hover test was performed, the drone suffered a deviation from its starting position of approximately $15cm$ and a rotational error of approximately 100^o clockwise. The latter is a matter of concern due to the fact that over an autonomous inventorying task, the drone will have completely misaligned itself with the environment around it. In an attempt to correct this problem, the drone's firmware was updated to the

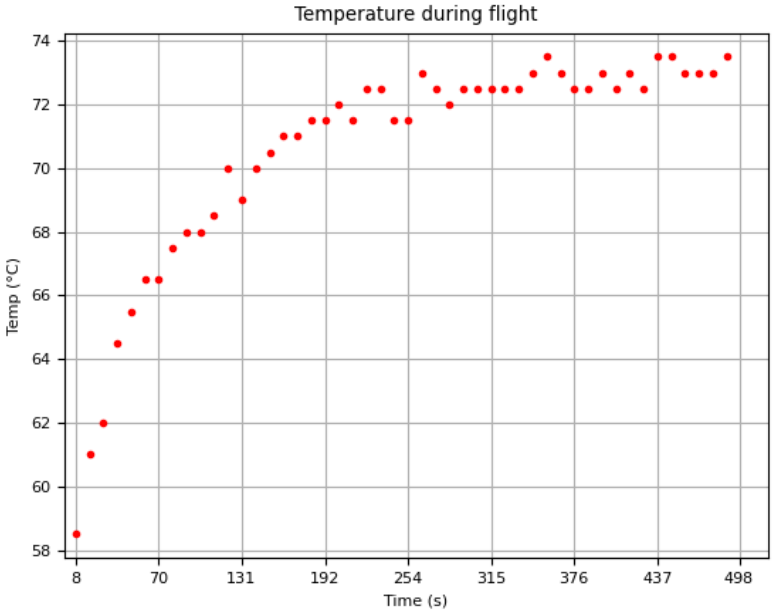

Figure 5.3: Drone internal temperature on stationary hover mode.

latest version.

Afterwards, by using the Tello Android application, two further calibrations were made: Firstly, the IMU Calibration to ensure the drone was calculating its position, attitude and heading correctly and secondly the COG (Center of Gravity) Calibration to balance the load of each of the four motors. The drone was tested a second time afterward the same way as the first. The results show neither the slight drift away from the starting position nor the rotation problem were satisfactorily resolved. After the 8-minute stationary hover test, the drone landed $14.76cm$ away from its original position and had rotated nearly approximately 80.6^{o} (a still severe heading error) clockwise from its original heading, as seen in Figure 5.4. The quadcopter blades showed signs of wear and were replaced.

Another stationary hover test, this time using the Tello Android application indicated the drift was still present. When each motor was inspected, it was found that the rear-right motor offered slightly more resistance to spinning the propeller than the others. This was verified by spinning each opposite pair of propellers respectively clockwise or counterclockwise (depending on their intended spinning direction). The rear-right propeller produced a different sound when spinning and its spinning decelerated at a faster rate. Finally, the drone was tested in a stationary hover flight. The starting orientation for each propeller was noted. The rear-right propeller was the last to start spinning while the remaining three started at the same time. When the drone landed, the same propeller was the first to stop (the remaining three stopped simultaneously at a later time) and the only one not to finish not to stop very close to its starting orientation. The conclusion to be derived from this situation is that the rear-right motor is either faulty or

more worn-out than the others. Fixing this issue was important since by having the drone yaw in place at a considerable rate, any commands made with an orthogonal layout in mind quickly became impossible for the drone to follow. For example, once issued a ”move” command for the drone to move forward $1.47m$, the drone would also drift right approximately $20cm$. A 90^{o} rotation would be heavily affected by the accumulated rotation the drone had done with its involuntary clockwise yawing, making it impossible to change to the second aisle consistently. Although not ideal, the solution for this problem involved experimenting with the average involuntary yaw the drone would pull per second. After several tests with the drone in a stationary hover, it was determined that the drone yawed clockwise on average $0.21^{o}/s$. This rate of involuntary yawing is not high enough to cause an update to the IMU's attitude values. Since rotation commands were already used, the simplest fix was to code a small threaded cycle to take into account the number of seconds passed since takeoff, and whenever the drone reached the end of an aisle, perform a second yaw correction. This correction would rotate the drone counterclockwise according to the following formula, where $TURN\_FIX$ is the angle to rotate counterclockwise, and t is the number of seconds that have passed since takeoff or since the last correction:

$$TURN\_FIX = 0.21t$$

After correcting the drone's yaw, $TURN\_FIX$ is added to the $OFFSET$ variable according to the following expression:

$$OFFSET\_NEW = OFFSET\_OLD + TURN\_FIX$$

This expression is required because even though the drone would be rotating to its ideal attitude, the IMU did not account for its involuntary yawing. Yet, the positioning correction command is issued and has an impact on the yaw value registered by the IMU since it accounts of a counterclockwise rotation of several degrees. Consequently, the $OFFSET$ variable is combined with the drone's current heading before determining the yaw correction required. This ensures the software stores the correct yaw values for the drone's skewed attitude values.

5.1.3 Unit Tests - RFID System

To begin simulating a warehouse environment, the physical limitations of the selected RFID hardware must be determined. The tag antenna is linearly polarized, therefore it and the RFID Reader's orientations are relevant.

Two experiments were made, described as follows:

1. An RFID tag (as seen on Section 4.1.2) was placed on a vertical, flat, non-metal surface. The tag is $7cm$ wide and $1,7cm$ tall and its depth is minimal. It was placed on the flat vertical surface with

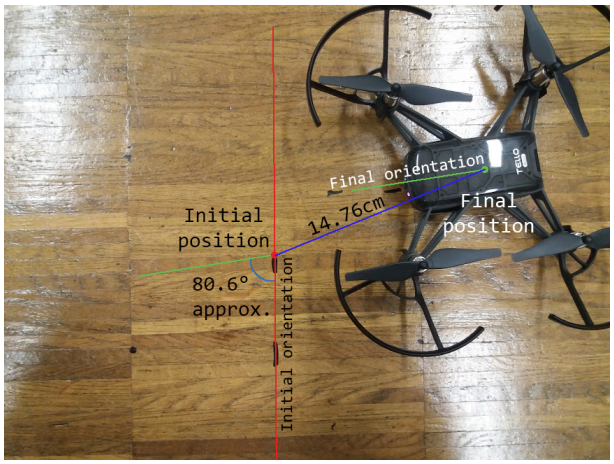

Figure 5.4: Comparison between drone takeoff position and orientation and landing position and orientation (top-down view)

its width as its base, parallel to the ground. The RFID Reader was placed vertically, with its frontal surface parallel to the tag's frontal surface, facing it. The RFID Reader can be tuned to several frequencies in the UHF range. This experiment verified the ideal SOAP for a tag, the choice of best frequency to use for the tags (by measuring the maximum read range for each frequency) and determined what the gain on the RFID Reader should be set to (by comparing three different power settings with their effective read range).

Tag SOAP The tag's SOAP specifications can be summarised as such:

- Size: The tag is $7cm$ wide, $1.7cm$ tall;
- Orientation: The tag is placed on a vertical surface with its widest sides parallel to the ground.
- Angle: The tag is read by the RFID Reader when the front-facing surfaces of both are parallel to each other.
- Placement: The tag is placed against a non-metal surface, in this case against the side of a cardboard box.

RFID Reader Frequency Setting The RFID Reader was set to its maximum gain. Afterwards, the RFID Reader was switched between each available frequency range (through the configuration application) while facing the RFID tag setup. For each frequency range, there was a distinct maximum read distance. The following table (Figure 5.5) presents the findings on the RFID Reader's maximum read distance per range.

The RFID Reader registered responses from the tag at significantly different ranges. Given that there was only a single tag being queried, collisions did not affect the tag's responses. This

Frequency Setting (MHz)	Experimental Read Distance
CN2 840~845	1.5cm
EU 865~868	1.5cm
US 902~928	22.5cm
CN 920~925	25cm
TW 922~928	25cm

Figure 5.5: Comparison between the available Query Frequencies and their Maximum Experimental Read Range

indicates the ideal settings to read the tag are the TW and CN frequency settings.

RFID Reader Maximum Read Range The power setting in the configuration application allows the RFID Reader to output a query signal with a gain of between approximately $-2dBm$ (Power Setting 0) to $25dBm$ (Power Setting 14), according to the tab's tooltip. Three power settings were tested: 0, 7, 14. The RFID Reader was configured to emit queries at the $922MHz$ to $928Mhz$ frequency range (TW frequency setting). The following table (Figure 5.6) presents the findings on the RFID Reader's maximum read distance with respect to gain.

Power Setting	Experimental Read Distance
0	7cm
7	15.5cm
14	25cm

Figure 5.6: Comparison between the minimum, intermediate and maximum power setting and the Maximum Read Range

The results obtained indicate maximizing the RFID Reader's gain when querying maximizes the distance at which it can read the tag.

2. Similarly to the first experiment, three RFID tags of the same make (Section 4.1.2) were placed on a vertical, flat, non-metal surface parallel to each other. First, they were placed at their tallest. Afterwards, they were placed at their widest. The RFID Reader was kept at a 20cm distance from the tags. It was configured to emit queries at the $922MHz$ to $928MHz$ frequency range (TW frequency setting) and its gain was set to its maximum value (power setting 14 at approximately $25dBm$). The RFID Reader's position was adjusted to be consistent with its placement in the first experiment and to always face the tags correctly as described by the Angle parameter of the tags' SOAP. The RFID Reader was then made to traverse the group of tags horizontally whilst maintaining the distance to them consistent.

Minimum Distance Between Tags It was found that when the tags were set parallel to each other at their tallest (with their narrowest $1.7cm$ side parallel to the ground), the RFID Reader

could collect at least one response from each tag when they were placed at least $8.5cm$ apart horizontally. When the tags were set with their widest side parallel to the ground, a minimum distance of $15cm$ was required to achieve the same result. This difference is in part due to the number of collisions in the tag responses to each of the RFID Reader's queries and due to the width of the tags themselves occupying much more of the RFID Reader's traversal range in the second iteration of this experiment.

To obtain experimental values while flying the drone at a constant speed of $20cm/s$ in a straight line, the RFID Reader was held below the drone, facing $20cm$ away from two series of tags and correctly oriented towards them: the first set standing at their tallest and the second at their widest. The experimental values for the distances at which to set the RFID tags apart from each other to ensure each tag was read were found to be larger:

- The tags at their tallest were only able to be all successfully read when set $15cm$ apart;
- The tags at their widest were only able to be all successfully read when set $20cm$ apart;

This can be summarized in Figure 5.7. We can conclude that ideally the tags should be placed at their tallest to maximize reading efficiency.

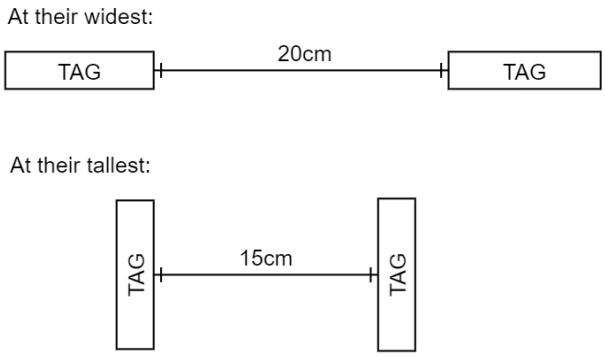

Figure 5.7: Ideal distance between tags, read from $20cm$ away.

5.1.4 Calibration - Distance to a visual marker and Computer Vision Shape Detection

The next step in building the warehouse simulation around the drone is related to its optical sensors. Once the drone's position in relation to a wall can be reliably configured, the space in which it will fly can be conceptualized.

Camera Viewing Angle According to the drone's specifications page webpage[2], the embedded camera has a Field-of-View of $82.6°$. Since this information does not specify the orientation of the Field-of-View, the following test was conducted to define the boundaries of the drone camera lens: Several points were drawn $10cm$ apart horizontally on a vertical wall to be used as guides for the camera boundaries. The drone was turned on along with its camera and connected to an Android phone with the Tello application. It was placed directly facing the set of points. The drone was then moved closer to the wall until two points $20cm$ apart were on the left and right boundary of the video stream. The same process was repeated for point pairs that were $40cm$ and $60cm$ apart. A right-angle triangle can be conceptualized from three points in space described as follows: the point where the drone camera is located (C); a point drawn on the wall (W); the halfway point between W and its pair (H). The following expression was applied to calculate the angle θ at $\angle HCW$ (named the horizontal viewing angle):

$$tan(\theta) = \frac{HW}{CW}$$

HW represents half the distance that separates the point pair. CW represents the camera's distance to the wall, obtained from experimentation, where the point pair is on the horizontal boundaries of the video output. The following table (Figure 5.8) presents the results obtained from applying the expression.

Point Pair Distance	Camera Distance to Wall	θ	2θ
20.0cm	17.3cm	30.11°	60.22°
40.0cm	37.0cm	28.36°	56.72°
60.0cm	58.2cm	27.24°	54.48°

Figure 5.8: Comparison between distance to the wall and horizontal viewing angle (2θ).

The test was repeated at the same distances from the wall, but with points separated vertically instead of horizontally. The 2θ values obtained from $tan(\theta) = \frac{HW}{CW}$ indicate that the camera's vertical viewing angle is 75% that of the horizontal viewing angle at a given distance from the wall. The accuracy of this ratio is verified by the described resolution of the camera (HD720p: specifically 960 by 720 pixels) for video output.

By comparing the results from each point pair, it can be concluded that the camera lens impacts the angle of the field of view at different distances from its target. As such, a value for the distance from the drone to the wall must be set before continuing. To ensure a minimum space for maneuvering, the drone will be set $60cm$ away from the wall.

Computer Vision Slider Calibration For the visual marker to be detected, a frame from the video feedback is first converted into one based on HSV (Hue, Saturation, Value) colour code. This new

[2]Tello EDU specifications: https://www.ryzerobotics.com/tello-edu/specs, 15 Dec 2020

image is then combined (through a bitwise ”AND” operation) with a mask representing the colour range that is to be selected. The result is a frame composed of black and white pixels, where white pixels represent the area where the selected colour range was detected. The colour range is flexible and can be adjusted for other colour ranges.

The resulting image is passed through a blurring bilateral filter to smooth the detected shape (covering small holes and correcting low-resolution blocks derived from the original frame) before shape detection takes place. To perform Canny edge detection[3] for the shape, one more value range (Threshold1 for the lower value and Threshold2 for the upper value) is used to serve as a cutoff for the color level between what will be considered black and white pixels in the blurred image.

To fully detect the visual markers used (the bounding rectangle lining up as close as possible with the actual shape of the visual marker), the ranges pictured in Figure 5.9 were used. It is important to note that lighting is a relevant factor when calibrating the drone in another environment. For reference, the room was being lit by tubular fluorescent ceiling lamps, with an approximate colour temperature of $5000K$.

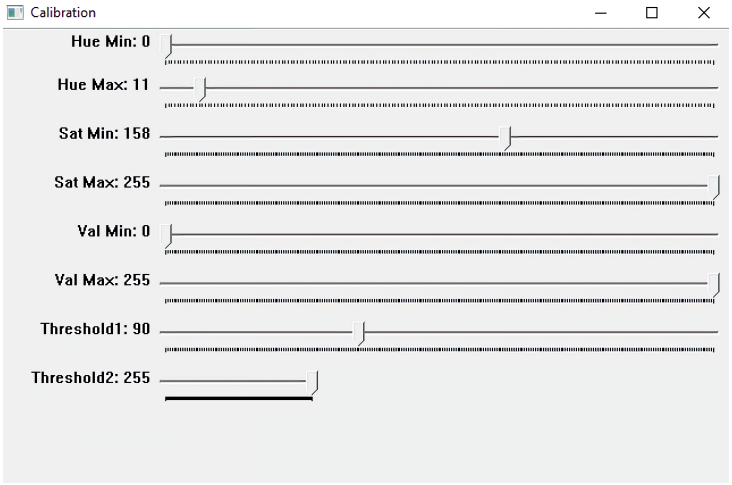

Figure 5.9: Slider settings for visual marker detection

Computer Vision Marker Identification Calibration Regarding identification, the slider calibration will detect the visual marker without issues. However, further testing was made to the conclusion that if the visual marker is in contact with the border of the video feedback, occasionally the shape detection will, instead of a rectangle, obtain a shape that encompasses exclusively the border of the visual marker. The centroid is still in an accurate position regarding the center of the visual marker, but the computed area of the shape is significantly inferior. This may cause issues: the drone might move forwards when this error occurs, causing it to move too close to the marker and crash against the wall.

[3]Canny Edge Detection: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html, 15 Dec 2020

It is important to note that the frames received from the drone's camera cannot be buffered, as that would cause the drone to not update its position in real-time, but instead according to outdated information (that would be the next frame after the one it just adjusted for). Whenever the module tasked with the computer vision logic fails to properly read only the last frame received (forced by keeping the buffer size to 1), the drone will be stuck correcting its trajectory until the software unfreezes or until it crashes. This problem was addressed by keeping only the latest frame in the video buffer and also by reducing the frames per second to 2 to reduce system load. However, this issue persists on occasion.

5.1.5 Calibration - Warehouse Mock-up

To simulate a warehouse environment, an area $4m$ wide, $5.60m$ long and over $2m$ tall in an artificially lit room was chosen. This area was divided as pictured in Figure 5.10. The shelf space was only delimited by plywood square panels $70cm$ in width which were covered by one large black bar and two bordering green bars to complement the behaviour of the VPS. This was to avoid crashing the test drone into the shelves in the event of an error. The visual markers for one shelf level were placed on the walls at a height of $1.40m$ from the floor. The warehouse layout consists of a grid of three nodes (x-axis) by four nodes (y-axis). The drone's starting position is the coordinate set (0,3). The drone will move along the aisles (along the y-axis) so as to scan the item tags. In this configuration, the drone reaches three Hybrid Checkpoint points (at (0,0), (1,3) and (2,0)) before ending its traversal.

To calibrate the drone's movement, its speed was set to $20cm/s$ and the "move" commands (in centimeters) were first issued in a 1:1 ratio with the measurements in the mock-up:

- Any x-axis movement was a multiple of $1m$ (the distance between two neighbouring nodes along the x-axis
- Any y-axis movement was a multiple of $1.47m$ (the distance between two neighbouring nodes along the y-axis)

This led to the drone crashing against the wall at the end of the aisle (cumulatively overshooting the goal of each movement) due to the drone not accounting for the time required to come to a halt. This issue was previously mentioned in Section 4.1.1.A and was much more noticeable than in smaller areas with more controlled movements. The proportionality for the drone's movements had to be estimated through trial-and-error and changes to its flight speed. The fine-tuning of the drone's horizontal movements remained proportional to the warehouse dimensions. The results indicated that the drone flying at $20cm/s$ would only require a "move" command for a distance of $85cm$ for x-axis movements and $120cm$ for y-axis movements pertaining to two neighbouring nodes. The irregularities in the drone's movements relate to its apparent disregard for the time it takes to accelerate and decelerate. Taking this into account, multiples of the resulting values remain relatively accurate. The drone may still either overshoot or not

reach its goal position if the distance is large enough. When calibrating this system for another layout, the distances to move between nodes along the x and y axes also needs to be adjusted from its intended values.

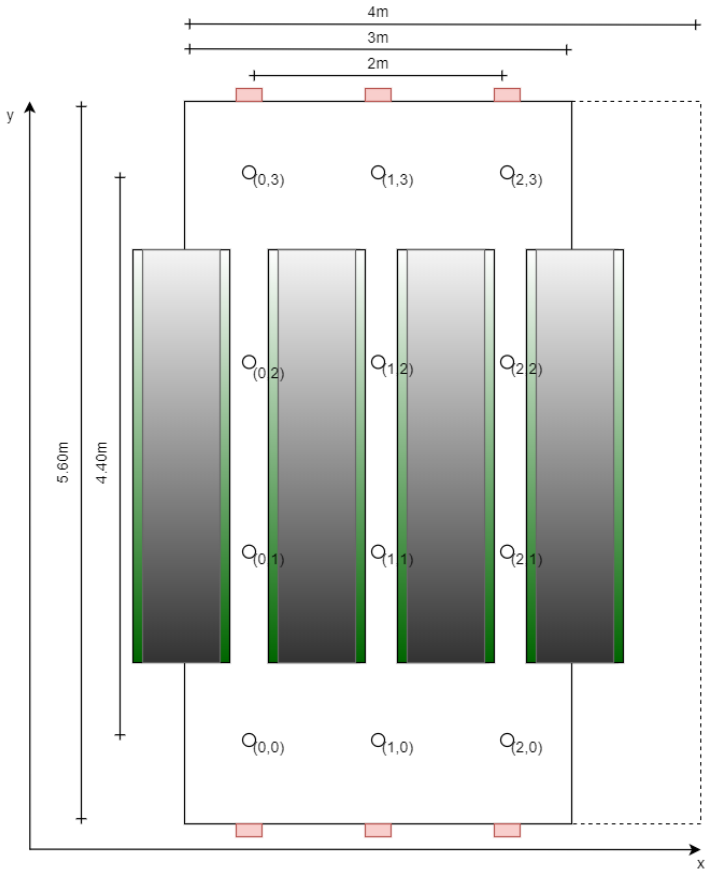

Figure 5.10: Warehouse Layout Mock-up. Each node is marked by a coordinate set. The floor panels are seen in black and green. The visual markers are placed on opposite walls.

5.2 Component Tests

5.2.1 Communications Component

The most important part of the communications component to test is its information flow. The test can be divided into two parts: the deployment of the RFID interface controller and the transmission of information. The local database was flushed for these tests since insertion times are a limiting factor.

RFID interface controller When the inventorying system is launched, the Wireshark instance that has created a new livebuffer.pcap file will start storing USB packets from the RFID Reader. Whenever a new entry is detected, the buffer file is read again. Depending on the amount of information, the time taken to

process new tag arrivals can vary. Of note is the fact that regardless of the time it takes to process each tag, Wireshark's configured filter will detect all tags that are successfully queried. This means all the tags in the buffer file will eventually be processed, even if the time it takes to complete the route through the warehouse is longer than the processing time.

To test the time to process tags, the system was started. Once it was ready, the following process was executed: **Scan a single blank tag; scan two item tags and one node tag in quick succession; scan an item tag; scan a node tag.** The results are as follows:

1. A single blank tag takes approximately 2 seconds to be processed;
2. The series of two item tags and one node tag takes approximately 4 seconds to be processed;
3. An item tag takes approximately 2 seconds to be processed;
4. A node tag also takes approximately 2 seconds to be processed.

The results indicate that the most costly part of the operation is not the amount of information inside a tag, but instead it is the time it takes for the whole cycle to read the buffer file to process any number of new tags.

Information flow To test the communications component's information flow management, the tag information from the previous test was used to determine if the previously empty local database and buffers were holding the correct information.

The results (respective to the previous test's points) are as follows:

1. The single blank tag was not present anywhere except for the Wireshark buffer file.
2. The local database's item table registered two item entries and the tag.txt buffer file contained the tag information from the first node tag;
3. The local database's item table registered one more item entry;
4. The tag.txt buffer file contained the tag information from the second node tag.

We can conclude the tags are being correctly parsed and being transmitted to the correct components (inventorying component for the item tags, navigation component for the node tags).

5.2.2 Inventorying Component

The test for the inventorying component relates to the identification of misplaced items and updating item counts. Two insertion queries were made to the local database, each for one item.

Two item tags were scanned: the first was correctly stored but had a smaller item count and the second was not in the right location but maintained its recorded item count.

1. The first item tag's item count value was updated. The item position value stored in the tag was checked against its entry in the local database and the flag value remained at 0 (indicating the item was correctly stored).

2. The second item tag's position values did not match the local database's records for that item. The second item's flag value was changed to 1 (indicating the item was misplaced).

The items were correctly updated to the local database, therefore the inventorying component is functioning as intended.

5.2.3 Navigation Component

5.2.3.A Virtual Coordinates Map Routing

The routing algorithm was tested to ensure the drone could be launched from any position and still be able to cover all the aisles it needed to. The algorithm accounts for shelves being arranged along the y-axis (vertically, according to Figure 5.11). The node map, seen in Figure 5.11 was purposefully made to be irregular to showcase the functionalities of the routing algorithm.

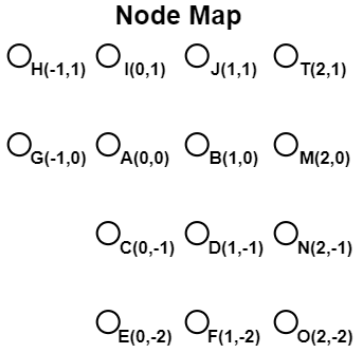

Figure 5.11: Node map used to test the routing algorithm.

The number of shelf levels was set to 3 (level 0, level 1 and level 2). The complete calculated route can be seen in Figure 5.12. The calculated route through the first shelf level explains a great deal about the drone's behaviour. The starting position was set on node A (0,0). The routing process follows:

- From the starting position, the shortest y-axis path to one of the edges of the warehouse is, in this case, a hop to I (0,1).
- The next hop will be C (0,-1) and then E (0,-2), since the goal is to traverse the length of each shelf.
- Afterwards, the next hop is towards another aisle: F (1,-2), D (1,-1), B (1,0), J (1,1).

- The next aisle follows: T (2,1), M (2,0), N (2,-1), O (2,-2).
- To move between O (2,-2) and G (-1,0), a diagonal movement is required (the movement control function issues a ”move up” command first so that the drone clears the maximum shelf height, moves the drone to the next hop, and then returns the drone to the previous altitude).
- Finally, the next hop is H (-1,1) (a command is then issued to move the drone to the next shelf level).

The remaining levels follow the same logic. Of note is the final hop of the final shelf level: the drone should return to its starting position. This leads the last hop, from G (-1,0) to A (0,0).

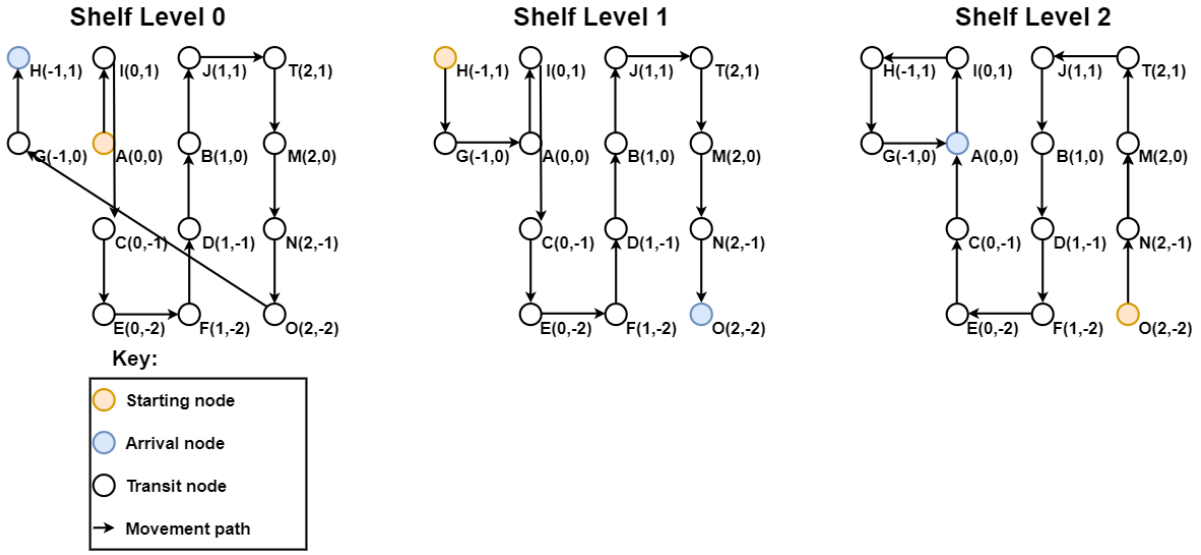

Figure 5.12: Routing a path through three shelf levels of the node map.

By Shelf Level 2, the drone would be following the ideal path for this layout: from O (2,-2) to G (-1,0), reserving it every other level. This path is the one with the least distance to be covered (excluding the final hop towards the original starting position in the example shown).

The impact this route has on the quantity and location of the Hybrid Checkpoint nodes to be used to correct the drone's position and trajectory is shown by Figure 5.13. On Shelf Level 0, the drone makes four stops to correct its course on a Hybrid Checkpoint (the correction on H (-1,1) is made after going up a shelf level). Even on Shelf Level 1, no stops are made at G (-1,0) due to the fact that it is not at one or the other extremes for the y-axis.

If the starting position was set to be on node O (2,-2), the number of hops would be minimized already on the first shelf level, causing there to be only four Hybrid Checkpoints to stop at per shelf level, the exception being Shelf Level 0 with only three stops, assuming the drone was started with its heading towards North (positive y-axis, default state).

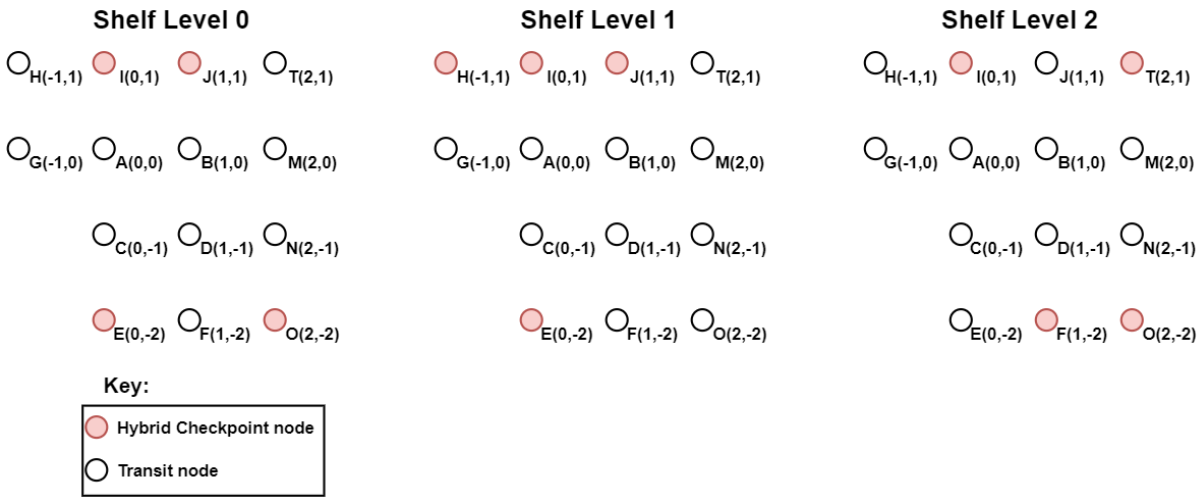

Figure 5.13: Hybrid Checkpoint nodes for the computed route in Shelf Level 0.

A more optimized starting position for the drone leads to leads to less distance to be covered and fewer stops to correct the drone's course, effectively increasing the drone's potential coverage.

5.2.3.B Mock-up System Test

To test the warehouse traversal capabilities of the drone, the warehouse layout mock-up, seen in Figure 5.10, was used. The navigation component was tested on a single shelf level. The plywood panels were removed to account for the impact of the VPS on navigation. The drone was set down on node D (0,3) and this node was defined as its starting position. The drone's starting heading was set to its default value of 0 (indicating it is facing towards the positive side of the y-axis). The warehouse mock-up can be translated into the node map seen in Figure 5.14.

System Test without Hybrid Checkpoints The drone was left to traverse the layout. Once the drone landed, the distance from its starting position to its landing position was measured (seen on the left in Figure 5.15). The drone traversed the warehouse layout in approximately 1 minute and 44 seconds. The accumulated error is significant: the drone landed approximately $1.57m$ away from the desired location (the x-component error and the y-component error being respectively approximate to $96cm$ and $127cm$).

System Test with Hybrid Checkpoints The exact same scenario was used for this test, with the exception that the Hybrid Checkpoint algorithm was used with a 30 seconds timer. Due to this last point, the drone traversed the warehouse in approximately 3 minutes and 54 seconds. This time, the drone landed only $0.56m$ away from its intended landing zone (seen on the right in Figure 5.15) (the x-component error and the y-component error being respectively approximate to $51cm$ and $25cm$). Comparatively, the

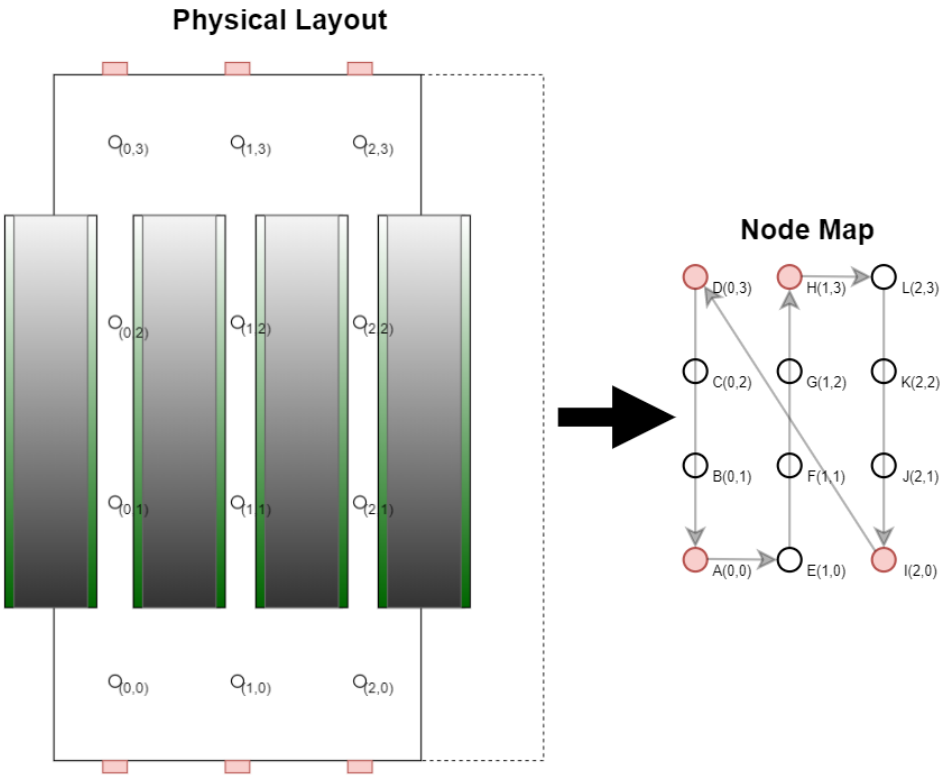

Figure 5.14: Physical to Virtual conversion of the warehouse space. The red nodes indicate the Hybrid Checkpoints for the first shelf level. The arrows indicate the route to be taken by the drone.

outcome of this test indicates that the developed Hybrid Checkpoint mechanism using computer vision improved the drone's accumulated positioning error by a significant amount.

5.3 System Field Test

To fully test the inventorying system, it is important to know how fast the drone can navigate the warehouse and the inventorying system's success rate in scanning a set of item tags. The main objective of the test is to measure how many tags were successfully read (translating into a success rate), how long was needed to traverse the warehouse layout, how close the drone's finishing position is to its starting position.

The same warehouse layout as the Hybrid Checkpoint one was used (Figure 5.14). A set of 13 tags were placed horizontally at between twenty centimeters and one and a half meters apart from each other at their tallest to represent items stored along two shelves: between x-coordinates 0 and 1; and between x-coordinates 1 and 2. Some tags were covered by a cardboard piece approximately $1cm$ thick. A single shelf level was accounted for when placing the tags. Node Tag verification was foregone to showcase the inventorying functionality more clearly. The drone is expected to stop for flight corrections four times,

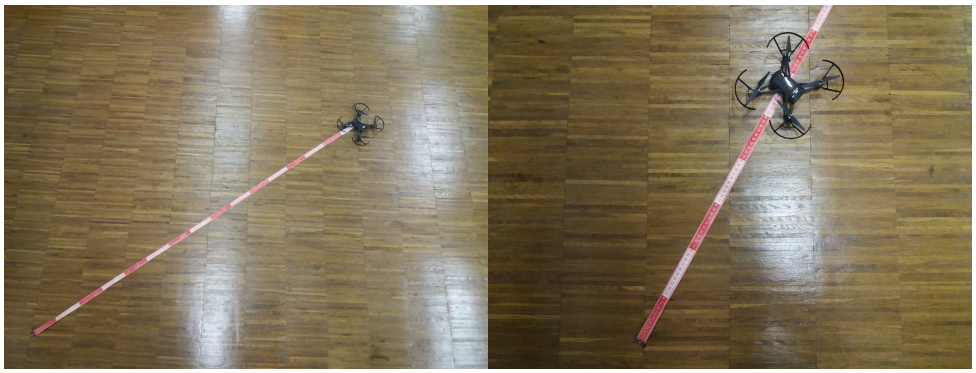

Figure 5.15: Drone flight test without (left) and with (right) Hybrid Checkpoints.

according to Figure 5.14. The drone is also expected to return as close to the starting position as possible. The user should be able to use the interface to see the updates to the local database's item table. The RFID Reader will be carried by hand along with the SBC (in this case, a laptop was used). To simulate the drone's ideal scanning capabilities, the RFID Reader will be moved in front of the tags at the same speed the drone is moving at. The Reader is expected to detect all tags separated by at least $20cm$ apart. Visual markers were placed at the end of every aisle at the same height. The computer vision algorithm is expected to use the drone's video feedback of the visual markers to correct the drone's position. The drone's yawing problem is also expected to be minimized over the course of the warehouse traversal.

The local database was populated beforehand as if it had received the file from the warehouse database. An excerpt of it can be seen in Figure 5.16. The example follows: In red, an item storage unit had 1 item removed from it and has been marked as misplaced by an on-site employee. Once the tag is scanned, the item count is updated on the local database and the item is marked as misplaced. In cyan, an item storage unit was restocked by 1 unit. Another prepared test is seen in Figure 5.17. The item storage unit, marked as correctly stored in the local database and on the item tag, is scanned. The position advertised by the item tag differs from the one stored in the records. The item's record is updated. This includes the item storage unit's current position and a flag to indicate the item's misplacement.

The local database contained records of 13 items (Figure 5.18). Of these items, two of the tags (henceforth named Overlap Tags) were less than $20cm$ apart and one tag (henceforth named Missing Tag) was present only in the warehouse records and not on the shelves. During the field test, 12 item tags were found. Upon further inspection, the only tag not found was the Missing Tag. This indicates an initial success rate of 100% in tag scanning according to the established limitations. To further test this value, the simulation was repeated five more times, but only over the Overlap Tags. Both Overlap Tags were successfully scanned only two times. Only one Overlap Tag was scanned another two times. No Overlap Tags were scanned one time. Although the field test's item tag scanning success rate was

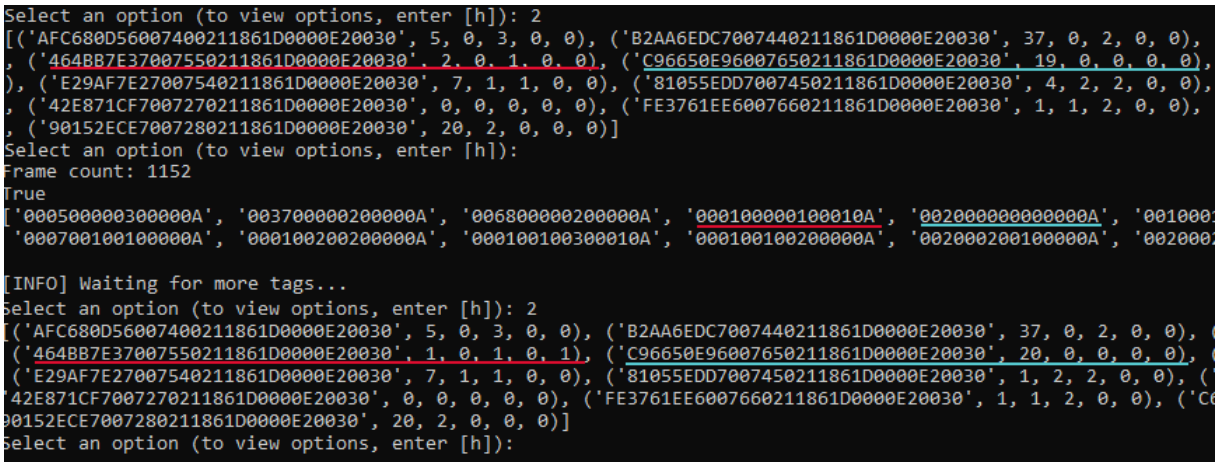

Figure 5.16: Output excerpt of the local database before and after field testing the inventorying system.

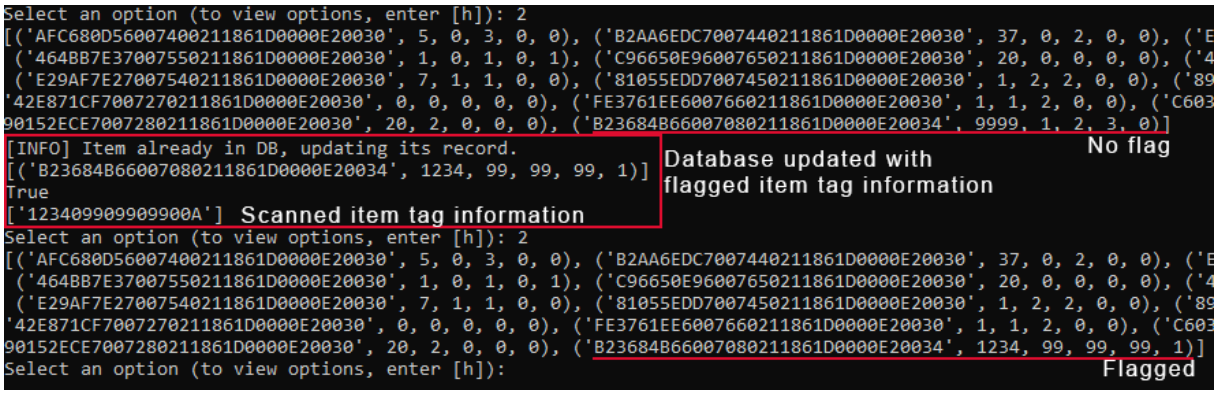

Figure 5.17: Output excerpt of flagging a previously unflagged tag due to conflicting item position attributes.

perfect, it is very likely that if Overlap Tags exist, the success rate will drop. In a worst case scenario, the number of successfully read tags will drop by the number of current Overlap Tags.

Overall Result The field test yielded mixed yet optimistic results. Each component individually executed its task correctly. When every part is combined however, the system suffers from setbacks related to the performance of the drone and RFID Reader. The result of the field test was an abrupt stop to the autonomous inventorying system. Unfortunately, the software-based fixes for the drone's involuntary yawing problem, described in Section 5.1.2, were not enough to curb it. Errors from the VPS further compounded upon its ongoing yaw error, which lead to even more extreme deviations from the path from the halfway point of the warehouse traversal onward. The drone had to be stopped when it reached node I (2,0) at the 5 minute and 48-second mark (which by the time the drone landed accounted for over 50% of its total battery-life), due to being roughly $2m$ off-course and already having over 45^{o} of uncorrected yaw error. The extent of the duration of the test was most likely what lead to this situation. If either the software-based corrections were more fine-tuned, or the drone did not suffer from its rear-right motor

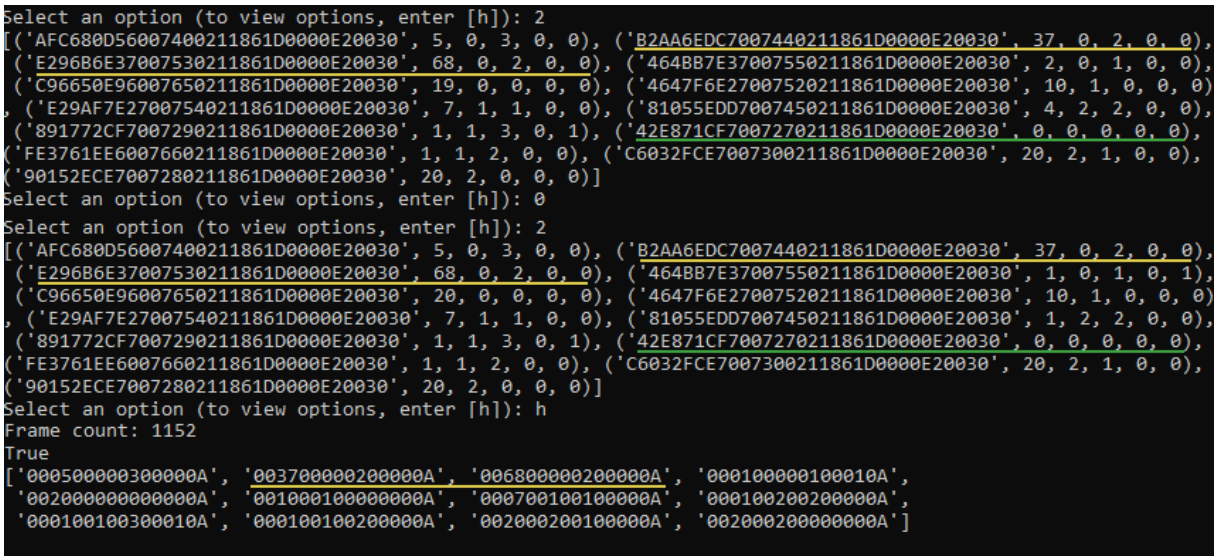

Figure 5.18: Querying the database before and after the field test. The Overlap Tags are shown in yellow. The Missing Tag is shown in green. The bottom list (taken from the output log) lists all the item tag information tag that was received.

issue in the first place, the final result would have been much more positive.

The RFID Reader was used $20cm$ away from the tags. The drone itself spans almost $20cm$. In a real-world scenario, the Reader would have to be much more powerful and almost certainly require bulk reading to obtain a moderate level of success in a cluttered environment with many more tags and metal surfaces to account for. The lack of bulk reading makes using the Reader to scan tags and validating the original proposal for an inventorying system much more cumbersome, although possible, as can be seen in the breakdown of the performance of each software component below. Finally, the fact the drone cannot carry all the required parts for the autonomous inventorying system is regrettable but this issue will be temporary, as drone manufacturers develop drones that can carry loads and possess larger, more efficient batteries.

Regarding the navigation component:

- The virtual coordinates map was obtained from a query to the local database and provided the correct route to traverse the warehouse (Figure 5.19).
- The commands to follow the planned route were issued to the drone;
- The computer vision algorithm obtained adequate corrections from the video feedback.

Regarding the inventorying and communications components:

- The local database was correctly accessed by the communications and inventorying components for selections, insertions and updates;

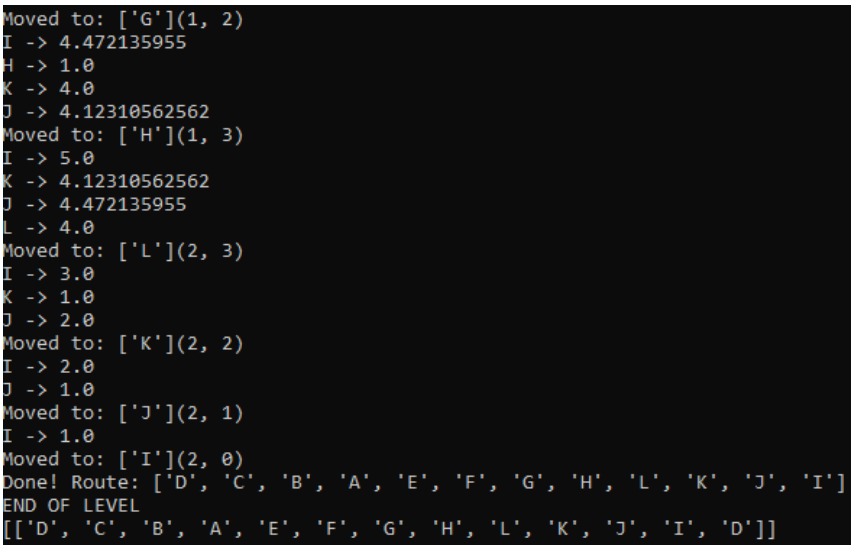

Figure 5.19: Excerpt of the output for the Virtual Coordinates Map. The computed route is shown at the bottom.

- The RFID interface controller correctly parsed the Wireshark capture file and obtained the scanned items' tag information;
- The inventorying component correctly updated the item tag information on the local database.

Virtual Coordinates Map and Drone Commands The Virtual Coordinates Map queried from the local database was used to correctly plot a course through the aisles. The outcome was exactly as expected, replicating the route from Figure 5.14. Every hop between two nodes was conducted by issuing rotation (when necessary) and movement commands to the drone.

Hybrid Checkpoints The drone's position was corrected at the Hybrid Checkpoints, notably at node A (0,0) where it lined up with the visual marker with a minimal margin of error. As the drone drifted off-course and its yaw error increased further down its route, it became difficult to rely on the computer vision algorithm to reliably correct the drone's positioning error at each Hybrid Checkpoint. Given the success of these initial corrections, if the drone was in full working order the Hybrid Checkpoints would prove to be a reliable way to correct the drone's trajectory.

Item Tag Processing By keeping up with the drone's movement and ensuring the tags were read by the RFID Reader at the correct speed, by the end of the field test, 12 tags had been scanned. The results follow: By comparing the first and second item table query in Figure 5.16, it is clear that the item counts of several tags were updated; those which were misplaced remained flagged; any tag that reported the wrong position when compared to its position in the local database was flagged as misplaced; of the Overlap Tags that were less than $20cm$ apart, the success rate was shown to fluctuate between

scanning none, one or both. Since no tags except the Missing Tag were missing in the buffer, it can be assumed that this setup does not require immediate line-of-sight to the tag, but previous testing shows that too many obstructions or metallic surfaces will reduce the RFID query's effective range. It can also be determined that the success rate for this setup was between 83.3% when considering the worst case scenario and 100% considering the best case scenario.

The field test results show the viability of the autonomous inventorying system for use in warehouse Reserve Areas and this work as a valid proof-of-concept. The current configuration is not ideal but can be improved upon to obtain greater results, namely by using an RFID Reader that is capable of bulk reading several tags omnidirectionally and a programmable drone capable in good working conditions capable of transporting the required equipment for this project. The field test, though not entirely successful, is a showcase of what is there to be improved and the culmination of the work, from where new projects can hopefully be developed.

5.3.1 Summary

Units tests were made to the drone and to the RFID Reader. The drone was tested in relation to its battery life, Signal-to-Noise Ratio and its internal chipset temperature. Battery-wise, the battery drops almost linearly when in flight, but lasts for 3 minutes less than expected. The Signal-to-Noise Ratio remains constant since the distance between the drone and the SBC remains constant. The drone does not possess internal cooling mechanisms temperature rises quickly and fluctuates between $70^{o}C$ and $75^{o}C$ as the drone overworks its motors to produce enough air current to cool itself. The drone's rear-right motor makes extended tests difficult due to the yaw skew it causes on the drone's trajectory. It is a large concern since the warehouse layout is strictly orthogonal. Corrections were attempted through the Tello application drone calibration functions. Unfortunately, neither the re-calibrations to the IMU nor to the Center of Gravity mitigated this issue. As a stopgap measure, a software-based solution was built: experiments were conducted to determine the drone's amount of yaw per second. Once an average value was obtained, the drone would be commanded to rotate counterclockwise by a multiple of that amount (determined by the time spent since the last correction) once it reached a Hybrid Checkpoint. The drone's IMU yaw value will become unduly skewed from this correction (but not from its involuntary yaw due its lack of IMU measurement resolution), therefore a global offset value must be updated to ensure the correct yaw value is queried from the drone's IMU.

The RFID Reader was expected to have a larger read range and tolerance to interference when signalling. While signalling the tags, the ideal frequency setting was found to be in the 922MHz to 928MHz band. The power setting was set to its maximum value ($25dBm$) to maximize its read range. The distance between tags should be minimized. Testing determined that if they are to be laid out at their tallest with respect to the ground, a minimum distance of $15cm$ should be ensured between RFID

tags so that they are read as the drone moves at a speed of $20cm/s$.

The autonomous inventorying system needs to be calibrated when it is used in a new environment. When attending to its calibration, the computer vision algorithm's detection was adjusted for the lighting conditions of the warehouse mock-up. Regarding the multipliers for the drone's movement, it was verified that the drone did not account for the time it needs to come to a halt. For larger distances and faster traversal speeds, the movement multipliers should be reduced to avoid overshooting the nodes or potentially crashing.

The software components function as expected. The communications component was tested by scanning a series of tags and measuring any significant processing time increases and it was found that the size of the buffer file was the most prominent factor. The tags were correctly parsed and transmitted to the direct components. The inventorying component was tested by inserting and updating item records into the local database. The items were correctly updated and the item tag made to signal a wrongly stored item was correctly flagged. The navigation component provided the shortest path route through all the nodes and shelf levels of a given purposefully irregular warehouse. The Hybrid Checkpoint approach was tested and the smart drone landed approximately $1m$ closer to its original starting position when utilizing this approach, compared to a landing position with an error margin of approximately $1.57m$ noted when testing the system without any course correction algorithms. It is also important to mention that if the video feed is frozen or otherwise not being transmitted for a longer period of time, the drone may perform incorrect adjustments based on outdated information.

The system field test was reliant on proper yaw corrections for the drone for an extended period of time. Even though each software component operated adequately and accurately (notably when adjusting the drone's position at the first Hybrid Checkpoint), the overall yaw skew from the drone's malfunctioning motor made it impossible to traverse the entire planned warehouse layout. The navigation component issued the correct commands and the computer vision algorithm correctly adjusted the drone's position whenever it stopped at a Hybrid Checkpoint. The drone's movement was manually simulated with the RFID Reader. The success rate for item tag scanning, according to the conditions of the field test and the limitations at the time, fluctuated between 83.3% and 100%, depending on whether there were overlapping tags (closer than $20cm$ apart) or not. The item tag information was always correctly updated.

6

Conclusions

Contents

Conclusions

6.1 Summary

6.1.1 State of the Art

Drones play an increasingly important role in improving efficiency across a multitude of tasks. This work focused on combining the mobility of a multicopter drone with existing technologies for scanning items to form an automated inventorying system capable of functioning within the Reserve Area of a warehouse. Such a system requires a dedicated navigation system for an indoor environment. Suggestions on what technologies to leverage were put forth: as for inventorying the items stocked in shelves, Passive RFID Scanning was chosen as it provides a simple and low-cost identification process for items. Comparatively to older barcode scanning technology, RFID scans can have longer ranges and do not require line-of-sight between the RFID Reader and the tags, although they are slightly more expensive. It depends on having each item or set of items labelled with a passive RFID tag which will have useful information written to it: the item's unique identifier, the item count of the storage unit, the position the item was stored at and a binary flag to determine if the item is misplaced or not. The inventorying system is ideally connected to one of the warehouse's database access points. A 2.4GHz Wireless LAN connection was the most direct choice, adding to the notion of near real-time updates to the warehouse's database. However, obstacles, interference and multi-path signal propagation reduce the connection's range and throughput. To remedy this, more access points can be deployed. Alternatively, RFly's solution, composed of using the drone as a relay platform for scanning tags from a centralized location in a large area, is a very interesting application of Radio Frequency signal properties. The Wi-Fi solution suffices since its application is direct and the infrastructure for its implementation is widespread. Regarding the drone's navigation of the warehouse, several paths may be taken: GPS seems like a clear solution, but it only works for outdoor areas. Warehouses are indoor environments and require a specialized solution. Between localization based on RSSI (be it through Wi-Fi, Bluetooth or RFID) and a solution based on UWB anchors, the more affordable RFID RSSI solution was chosen, which also implies the same RFID Reader will be used for both inventorying and navigation functionalities. There are

limiting physical constraints regarding the equipment choices at first glance. The main limiting factors relating to the drone are its maximum flight time (limited by its battery-life) and its payload capacity (its ability to transport the required sensors). Cluttered or otherwise metallic surfaces interfere with the RFID Reader's scanning capabilities. Legal issues arise with the use of drones in spaces where employees work. Precautions must be taken to ensure the work environment is safe and that no laws regarding UAV flight or surveillance are broken. Other related works (apart from the above mentioned RFly solution), such as RFIDRead's use of drones in a storage yard scenario with RFID inventorying technology by Bae, Sung Moon, et al, or the UWB localization solution for drones by Macoir, Nicola, et al, or even the commercially available solution by PINC AIR served as inspiration and a source of knowledge for this project.

6.1.2 Architecture

The inventorying system can be built according to a set of defining parameters, such as: the multi-copter **drone's characteristics** relating to its battery-life, its available movement options and behaviour and safety precautions; the **warehouse's physical infrastructure**, its indoor layout in the Reserve Area and the item storage method; the **Warehouse Telecommunications Infrastructure**, such as the warehouse's Wireless LAN network and its centralized item database; **item identification and location methods** relying on reusable tags; and the possibility to **navigate the warehouse** according to its layout.

The inventorying system was expected to be able to scan item tags located throughout a warehouse Reserve Area of varying sizes. These items could be obstructed or not in line-of-sight. These reusable item tags should contain information that allows the inventorying system to keep the warehouse database updated in near real-time and fag misplaced items in such a way that it allows the warehouse to remain operational. The system should be autonomous. The drone transporting the equipment is tasked with navigating the warehouse and correcting its trajectory in the case it veers off-course. The system is planned to contain a local database in which the smart drone stores item tag information to be sent to the warehouse database whenever a loss-tolerant connection to one of its access points is available. The route should be available when the drone takes off. An inventorying system interface should exist to ensure the warehouse layout is configurable and the local database can be accessed.

This set of parameters and requirements sets the scope of the project. The smart drone (the autonomous inventorying system) routes and navigates a path through an orthogonal warehouse environment of varying dimensions (programmable by the user), moving along its aisles and scanning items. The drone scans each shelf level horizontally until the entire Reserve Area has been traversed. Whenever it can connect to a warehouse access point, it will update the warehouse database through a loss-tolerant communications protocol. The smart drone is able to adjust its position to ensure it remains in

the correct path. The drone, once finished, plots a diagonal path back above the maximum shelf height to where it started and lands. The smart drone can be subdivided into its item sensing and warehouse navigation systems. The respectively named **inventorying component** and **navigation component** are brought together by the **communications component**, tasked with information parsing and delivery as well as providing a user interface, and by the locally stored **database component**, which contains the information required for both the inventorying and navigation components.

The **inventorying component** takes the information from the scanned item tags and parses it to obtain the EPC, the item count, and the item position. When it updates the local database with this information, it checks the item's position against the locally stored item position. If they differ, the component adds a binary flag value to the item tag information to mark it as a misplaced item if it had not yet been flagged. Otherwise, if it was already flagged it remains so. An RFID sensor system (a reader and a set of tags) compatible with EPC Class 1 Gen 2 should be able to store an item's EPC, count, position and optional flag). Ideally, the RFID Reader should be capable not only of **signalling** individual tags but also of **bulk reading** several tags at once. Tag SOAP indications should be respected to maximize scanning effectiveness.

The **navigation component** uses a **Virtual Coordinates Map** to plot its route through the warehouse. This virtual coordinates map converts the physical environment into a digital one according to graph theory. The graph nodes represent intersections between corridors and the lines indicate available paths. The routing algorithm seeks to find the shortest path between the node the smart drone is currently on and the next one in the aisle it is in. The smart drone seeks to traverse the warehouse by using the least amount of movement possible, which leads it to traverse an entire aisle before shifting over to the next one. The end result is a single branch spanning tree that passes through all nodes once. Both to complement this routing algorithm and to bridge the gap between the physical and virtual world, two navigation approaches were detailed:

- The **RSSI Triangulation approach** relies on the RFID Reader's bulk reading capabilities to query several node tags and obtain an RSSI value from each, from which it calculates an approximation of its real-world position. This would be complemented by information regarding the drone's previous movement and its reported previous position. Node tag placement favours redundancy to ensure a minimum of three RFID node tags are detected. The drone's computed position is then used to perform any corrections to its trajectory it requires or to adjust its route.

- The **Hybrid Checkpoint approach** uses a combination of RFID node tags and a set of visual markers on the walls opposite each other at the end of each aisle. By leveraging the drone's front-facing camera and computer vision technology, the smart drone can correct its position whenever it reaches the extremity of an aisle. It can then verify its position by scanning the node tag placed next to the visual marker. This approach requires the use of existing open-source libraries to parse

the video feedback.

Much like the inventorying component, both approaches benefit from adequate tag SOAP indications.

The **database component** stores two tables:

- The Item Table holds the following attributes: **EPC** (Primary Key); **Item count**; **x-axis Position**; **y-axis Position**; **z-axis Position**; **Flag**.
- The Node Table holds the following attributes: **EPC**; **x-axis Position**; **y-axis Position**.

The tables have no relation between them. Each can be accessed respectively by the inventorying component and the navigation component. The communications component accesses both.

6.1.3 Implementation

The whole inventorying system can be separated into five parts: the **Smart Drone**, which contains the multicopter, the SBC and the RFID Reader; the **Item and Node RFID tags**, the **Visual Markers**; the **Network** connections from the SBC to the drone and from the SBC to the warehouse database; and the **Software architecture**.

The drone is a DJI Tello EDU, a small $80g$ multicopter programmable in Python through a provided SDK. It generates a 2.4Ghz Wireless LAN which will be used to receive commands sent by the computational platform. The drone is equipped with a series of sensors: an IMU, a VPS, and a frontal camera. The IMU lacks floating point resolution. The VPS is very sensitive to changes in lighting and ground surface conditions. When in flight, the drone can move along all three-dimensional axes, yaw and adjust its speed. These functionalities are sufficient to navigate the warehouse. The drone suffers from an issue on its read-right motor that causes it to involuntarily yaw clockwise. When attempting to fix this issue via a software-driven yaw correction system, it was noted that it was difficult to obtain accurate adjustment values since the drone yawed at differing rates accross different experiments (the adjustment value is based on the average yaw skew per second). This problem is compounded by the fact the IMU does not register the relatively small $(0.208^{o}/s)$ involuntary yaw but does in fact register the comparatively much larger yaw correction made whenever possible. The corrections made work best for shorter flight times.

The RFID Reader is a WRD-130-U1 Ultra High Frequency RFID card reader operating in the 860Mhz to 960Mhz frequency range. It scans tags through signalling and sends the information received through keyboard emulation functions. It is not capable of bulk reading and is heavier than the drone. It is therefore not possible to operate the drone with the required equipment. The navigation positioning adjustments to be taken will necessarily follow the **Hybrid Checkpoint** approach. The RFID Reader was difficult to operate at first not only due to unfamiliarity with the technology but also due to the lack of useful and precise documentation available to detail its behaviour. The RFID Reader was configured to

read and write 32 EPC hexadecimal digits and 16 characters of tag information on the RFID tags which were acquired.

The aggregating element for the smart drone is the SBC. In particular, it is suggested to use a Raspberry Pi Zero and in this case a TP-Link TL-WN321G 54 Mb/s Wireless USB Adapter to allow for the connection from the SBC to the warehouse database. However, since the hardware architecture would require it to also be transported by the drone, it was replaced by a laptop of increased performance to facilitate development.

The RFID tags that were used were LN-9662, Higgs-3, "Short" Inlay type tags. These comply with the EPC Class 1 Gen 2 protocol and were divided into a set of Item tags and a set of Node tags, wherein the respective tag information was written to.

Field testing will require the drone to fly the intended path and the computational platform to be operated manually to simulate the intended load carrying capacity of the drone. The inventorying component will maintain its original functionalities, limited only by the RFID Reader's lack of bulk reading function. The tags will have to be placed on the items at the same height level to ensure the smart drone can scan them while traversing the aisle in a single horizontal movement. The navigation component will forcibly be the **Hybrid Checkpoint approach**. This is an opportunity to utilize current open-source computer vision libraries to assist with the drone movement. Signalling will be limited to verifying the drone's position once it finishes adjusting its position through the camera positioning algorithm. This algorithm will use a set of bright red visual markers placed on the walls at each end of the aisles to provide cues for the drone to adjust its position. The networks required for this project are two Wi-Fi connections. The first connects the SBC to the drone. The second (through the external Wi-Fi adapter) connects the SBC to the warehouse database access points.

The code architecture was developed in Python. In particular, Python 2 was used for the navigation component due to the reliance of the drone SDK on this version. The rest of the system will operate under Python 3. All three software components access the database through the slite3 library to read, write and update information where required. To allow for communication between the two Python versions, a tag.txt buffer file is used since both can access the file without issues regarding serialization and de-serialization of objects. The item and node tag information is captured by a Wireshark filter (which writes to a livebuffer.pcap buffer file and accessed actively through PyShark. It is then parsed and sent respectively to the inventorying component or to the navigation component. The local database stores an item table and a node table, both configurable through the communications component. The communications component uses the application layer File Transfer Protocol to update the warehouse database. The navigation component computes the route to be followed by the drone through the Virtual Coordinates Map. After takeoff, it is tasked with issuing the commands to move the drone through the route. The navigation component uses computer vision to, when the smart drone reaches an edge of

an aisle, detect the aforementioned visual marker. It then computes the centroid of the shape and its area. By measuring the distance and direction of the vector from the centroid of the shape to the center of the video feedback, its vertical and horizontal components can be obtained. These result in ”move” commands issued to the drone, respectively ”up” or ”down”, and ”left” or ”right”. Lastly, the area of the shape is used to determine the the drone needs to approach or move away from the visual marker (through ”move forward” or ”move back” commands).

6.2 Discussion

The outlook after development is an optimistic one. The goal to envision and create a proof-of-concept for an autonomous inventorying system was met, albeit with caveats. Notably, the main issue relates to the inadequate material used to develop this project. This was mainly to due inexperience on the subjects of RFID and drone technologies. Selecting the equipment should have been made with guidance on the subject. The research previously made on these technologies fell short of detailing the amount of specialization they have, and so a significant part of the work consisted of developing workarounds for problems and flaws discovered throughout development and adjusting the initial system design to suit them.

The learning process involved first understanding how the drone operated. By understanding how to pilot it and how to connect to it via the SDK, operating it became a matter of grasping the limitations of the drone. It is important to note that the faulty drone was the main cause for the incomplete field test. This issue was difficult to pinpoint and it is still unclear if the cause was a factory defect or post-purchase damage. Android application re-calibrations did not correct the issue. In an attempt to fix this problem without recalling the drone, a software solution was built upon experimental data to correct the drone’s yaw according to information polled from its IMU. The IMU also assisted in developing the Hybrid Checkpoint solution. Both of these efforts were limited by the IMU’s lack of precision. This point is critical, since if the IMU was sensitive enough, it would also pick up its involuntary rotation and possibly even correct itself automatically. If this was the case, the final outcome would see a much cleaner flight through the warehouse. The drone’s movements remain imperfect, so the IMU is used to provide corrections to the involuntary clockwise yaw problem and to slight imperfections in rotation commands, such as rotating the drone with a margin of error of within a few degrees. The drone’s VPS was sensitive to the point that it worked as a detriment to it. Although it functioned properly when responding to an imbalance or shift in surface height below it, the VPS also caused the drone to stray off-course or even crash due to misjudging its surroundings due to lighting or floor conditions. However, even with these issues, it was possible to prove the basic theory of the work: that drones can be used indoors, provided they operate not only through a virtual map but also through receiving and processing feedback from the

physical world. Even though the drone is faulty and cannot carry the necessary equipment, the fact that it is programmable is what made this project feasible.

The selected RFID parts should have supported bulk reading for this project. Given that it is a mature technology, the limited showcase of their functionality is still a strong indicator it is a good match for an automated system. It was shown to be accurate and consistent in its results regarding scanning. Although the expected read range was much shorter than the advertised one, it does not invalidate the use-case of the inventorying system. It is not without its flaws, namely the reduced range from tag SOAP and metallic interference which is certainly common in warehouse shelving. The information stored in the programmable tags exemplifies the usefulness of RFID tags over other widespread identification methods. The relatively large distance to be kept between them can be minimized by a bulk reading and with it increasing query rates. Collisions can be avoided through the use of the SBCS protocol, for example. The recommendation is to mainly consider a lightweight RFID Reader clearly capable of far-field bulk reading.

The implementation of computer vision as a means to perceive the world through everyday optical sensors helps demonstrate the various avenues for developing a project such as this. With responsive equipment, OpenCV provides accurate, useful and reliable information when its use-case is adequately compartmentalized. Its performance was adequate and mostly flawless (save for issues regarding the processing time for frames, backlogged frames and wireless connection problems) for the type of quick real-time adjustments made to the drone during flight.

Besides acquiring or developing dedicated equipment for a task such as this, the main points to improve on would be developing a robust and lightweight RFID scanning method taking into account the options described throughout this work, particularly in the State of the Art section. The computer vision algorithm would also benefit from improvements regarding more efficient video processing to minimize issues with processing time and lost or otherwise unprocessed frames. This system could be expanded beyond the Hybrid Checkpoint approach and take a more active role in helping the drone navigate the warehouse.

6.3 Future Work

This work serves as an initial proof-of-concept to help demonstrate the viability of using drones to automate to operate in indoor environments to perform tasks otherwise laborious attributed to human employees. The drone industry is growing and developing ever more capable and cost-effective drones. The current state of the project suggests that, with equipment better suited for the scenario, one could develop an effective solution for minimizing warehouse inventorying downtime and allow for near real-time updates on stock for e-commerce, shipping and storage platforms. The autonomous inventorying

system is a foundation on which to build upon with further work and optimizations. An interesting proposition is to develop this idea towards the vision of having a smart drone inventorying system product with minimal setup and rapid deployment. To summarize, the project was shown the be a solid starting step on which to develop further work on developing an efficient autonomous inventorying system for warehouses with minimal human interaction, as we draw closer to automating the logistics chain from production to consumption.

Bibliography

[1] P. Lou, Q. Liu, Z. Zhou, and H. Wang, "Agile supply chain management over the internet of things," in *2011 International Conference on Management and Service Science*. IEEE, 2011, pp. 1–4.

[2] L. McCathie, "The advantages and disadvantages of barcodes and radio frequency identification in supply chain management," 2004.

[3] M. P. Groover, *Automation, production systems, and computer-integrated manufacturing*. Pearson Education India, 2016.

[4] N. Macoir, J. Bauwens, B. Jooris, B. Van Herbruggen, J. Rossey, J. Hoebeke, and E. De Poorter, "Uwb localization with battery-powered wireless backbone for drone-based inventory management," *Sensors*, vol. 19, no. 3, p. 467, 2019.

[5] J. Gu, M. Goetschalckx, and L. F. McGinnis, "Research on warehouse operation: A comprehensive review," *European journal of operational research*, vol. 177, no. 1, pp. 1–21, 2007.

[6] B. Rouwenhorst, B. Reuter, V. Stockrahm, G.-J. van Houtum, R. Mantel, and W. H. Zijm, "Warehouse design and control: Framework and literature review," *European journal of operational research*, vol. 122, no. 3, pp. 515–533, 2000.

[7] "Latest version of the smart drone autonomous inventorying system," https://github.com/D-K-O-I/thesis-autonomous-inventorying-system, accessed: 29-12-2020.

[8] "Amazon prime air showcase," https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011, accessed: 07-12-2019.

[9] A. Welch, "A cost-benefit analysis of amazon prime air," 2015.

[10] L. Xu, V. R. Kamat, and C. C. Menassa, "Automatic extraction of 1d barcodes from video scans for drone-assisted inventory management in warehousing applications," *International Journal of Logistics Research and Applications*, vol. 21, no. 3, pp. 243–258, 2018.

[11] Y. Ma, N. Selby, and F. Adib, “Drone relays for battery-free networks,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 335–347.

[12] D. Dressen, “Considerations for rfid technology selection,” *Atmel Applications Journal*, vol. 3, pp. 45–47, 2004.

[13] E. F. Golen, N. Shenoy, and X. Cao, “A low latency scheme for bulk rfid tag reading,” in *2008 IEEE Wireless Communications and Networking Conference*. IEEE, 2008, pp. 1565–1569.

[14] A. Ustundag, S. Kilinc, and O. Kabadurmus, “Evaluation of operational parameters affecting bulk reading performance of uhf rfid system,” in *2007 1st Annual RFID Eurasia*. IEEE, 2007, pp. 1–4.

[15] M. Asadpour, D. Giustiniano, and K. A. Hummel, “From ground to aerial communication: Dissecting wlan 802.11 n for the drones,” in *Proceedings of the 8th ACM international workshop on Wireless network testbeds, experimental evaluation & characterization*, 2013, pp. 25–32.

[16] S. M. Bae, K. H. Han, C. N. Cha, and H. Y. Lee, “Development of inventory checking system based on uav and rfid in open storage yard,” in *2016 International Conference on Information Science and Security (ICISS)*. IEEE, 2016, pp. 1–2.

[17] K. Srinivasan and P. Levis, “Rssi is under appreciated,” in *Proceedings of the third workshop on embedded networked sensors (EmNets)*, vol. 2006. Cambridge, MA, USA., 2006.

[18] A. Bose and C. H. Foh, “A practical path loss model for indoor wifi positioning enhancement,” in *2007 6th International Conference on Information, Communications & Signal Processing*. IEEE, 2007, pp. 1–5.

[19] J. Hallberg, M. Nilsson, and K. Synnes, “Positioning with bluetooth,” in *10th International Conference on Telecommunications, 2003. ICT 2003.*, vol. 2. IEEE, 2003, pp. 954–958.

[20] Y. Wang, X. Yang, Y. Zhao, Y. Liu, and L. Cuthbert, “Bluetooth positioning using rssi and triangulation methods,” in *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. IEEE, 2013, pp. 837–842.

[21] S. S. Saab and Z. S. Nakad, “A standalone rfid indoor positioning system using passive tags,” *IEEE Transactions on Industrial Electronics*, vol. 58, no. 5, pp. 1961–1970, 2010.

[22] Z. Tinqting, Z. Qinyu, Z. Naiionq, and X. Hongguang, “Performance analysis of an indoor uwb ranging system,” *Journal of Systems Engineering and Electronics*, vol. 20, no. 3, pp. 450–456, 2009.

[23] K. Witrisal, S. Hinteregger, J. Kulmer, E. Leitinger, and P. Meissner, "High-accuracy positioning for indoor applications: Rfid, uwb, 5g, and beyond," in *2016 IEEE International Conference on RFID (RFID)*. IEEE, 2016, pp. 1–7.

[24] A. Anand, S. Agrawal, S. Agrawal, A. Chandra, and K. Deshmukh, "Grid-based localization stack for inspection drones towards automation of large scale warehouse systems," *arXiv preprint arXiv:1906.01299*, 2019.

[25] I. Derpich, D. Miranda, and J. Sepulveda, "Using drones in a warehouse with minimum energy consumption," in *2018 7th International Conference on Computers Communications and Control (ICCCC)*. IEEE, 2018, pp. 97–102.

[26] "Current version (1.1) of decawave's application note aps011," https://www.decawave.com/wp-content/uploads/2018/10/APS011{_}Sources-of-Error-in-Two-Way-Ranging-Schemes{_}v1.1.pdf, accessed: 02-12-2020.

[27] "Pinc air," https://www.pinc.com/warehouse-drone-inventory-management/, accessed: 22-10-2019.

[28] "Skyrfid inc., mid-range technologies," https://skyrfid.com/Mid-Range_RFID.php, accessed: 05-01-2020.

[29] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[30] S. Pettie and V. Ramachandran, "An optimal minimum spanning tree algorithm," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2000, pp. 49–60.