Report on Warehouse Drone Collision Avoidance

Bachelor Thesis

Submitted by Tristan van Vegchel

In fulfillment of the requirements for
the degree
Bachelor of Science in Informatics
To be awarded by the
Fontys Hogeschool Techniek en Logistiek

Venlo, January 13, 2020

Information Page Internship / Graduation Report
Fontys University of Applied Sciences
School of Technology and Logistics
Post Office Box 141, 5900 AC Venlo, Netherlands

Type of report:	Graduation report
Student name:	Tristan van Vegchel
Student number:	2762870
Study:	Software Engineering
Period:	September 2019 – February 2020
Company name:	Seacon Logistics
Address:	Celciusweg 60
Postal code + City:	5928 PR Venlo
Country:	The Netherlands
Telephone:	+31 (0)77 - 327 55 55
Company supervisor:	Kai-Arne Reiter
Supervising Lecturer:	Sander Bruinsma
External commissioner:	T.B.D.
Company Confidential:	*No*
Number of words:	6224
Coding:	*(see document "Coding of the Research". Logistic students only)*

The warehouse drone collision avoidance project concerns itself with the analysis, design, and development of a solution to provide a drone with collision avoidance functionality. The time scope was set for 1 semester (5 months), and resulted in the creation of this bachelor thesis. This project contains 2 prototype concepts: A concept based on reinforcement learning and a concept based on combining computer vision with a state machine implementation. The reinforcement learning concept did not show promising results as it did not show any sign of learning. The state machine concept its components work separately, but still requires smoother transitioning and a number of bugs needed to be fixed before it is able to run the entire state machine properly. This project also contained a research aspect. This research focused on the viability of Artificial Intelligence techniques for collision avoidance. Out of this came 3 approaches, of which one was created a prototype for. From the research it can be concluded that a single, stand-alone reinforcement learning algorithm is insufficient when attempting to use drones for cycle counting.

STATEMENT OF AUTHENTICITY

Issued by the FHTenL Examination Board, September 2017

I, the undersigned, hereby certify that I have compiled and written this document and the underlying work / pieces of work without assistance from anyone except the specifically assigned academic supervisor. This work is solely my own, and I am solely responsible for the content, organization, and making of this document and the underlying work / pieces of work.

I hereby acknowledge that I have read the instructions for preparation and submission of documents / pieces of work provided by my course / my academic institution, and I understand that this document and the underlying pieces of work will not be accepted for evaluation or for the award of academic credits if it is determined that they have not been prepared in compliance with those instructions and this statement of authenticity.

I further certify that I did not commit plagiarism, did neither take over nor paraphrase (digital or printed, translated or original) material (e.g. ideas, data, pieces of text, figures, diagrams, tables, recordings, videos, code, ...) produced by others without correct and complete citation and correct and complete reference of the source(s). I understand that this document and the underlying work / pieces of work will not be accepted for evaluation or for the award of academic credits if it is determined that they embody plagiarism.

Name (in capital letters):	TRISTAN VAN VEGCHEL
Student number:	2762870
Place / Date:	Venlo / 13-01-2020
Signature:	[illegible]

Contents

List of Figures

List of Tables

List of Listings

Revision History

Revision	**Date**	**Author(s)**	**Description**
0.1	24-10-2019	Tristan van Vegchel	Created skeleton
0.2	15-11-2019	Tristan van Vegchel	Added introduction
0.3	31-12-2019	Tristan van Vegchel	Added SRS
0.4	03-01-2020	Tristan van Vegchel	Added approaches
0.5	06-01-2020	Tristan van Vegchel	Added project description
0.6	08-01-2020	Tristan van Vegchel	Added state machine concept
0.7	09-01-2020	Tristan van Vegchel	Updated report after feedback
0.8	10-01-2020	Tristan van Vegchel	Added reinforcement learning concept
0.9	12-01-2020	Tristan van Vegchel	Added conclusion & recommendations
1.0	12-01-2020	Tristan van Vegchel	Second review and completion of first version
1.1	13-01-2020	Tristan van Vegchel	Consistency changes & added drone and state descriptions
1.2	13-01-2020	Tristan van Vegchel	Added appendix for installation and setup.
1.3	13-01-2020	Tristan van Vegchel	Grammar changes after third review

List of Abbreviations

A.I. Artificial Intelligence.

CBS Centraal Bureau voor de Statistiek [Statistics Netherlands].

DQN Deep Q Network.

Fontys Fontys University of Applied Sciences.

IL Imitation Learning.

MIT Massachusetts Institute of Technology.

ML-Agents Machine Learning Agents.

PID Proportional-Integral-Derivative.

RACI matrix Responsible-Accountable-Consulted-Informed matrix.

R-CNN Region-based Convolutional Neural Network.

RL Reinforcement Learning.

SRS Software Requirements Specification.

SSD Single-Shot Detector.

STG2 Graduation Project.

Definitions

Artificial Intelligence The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages (Lexico, n.d.).

Cycle count Inventory control & management process in which a (small) subset of the inventory in a warehouse in a specific location is counted on a specific day. This is often done daily, with a new section done every day until the whole inventory has been counted. It is done in order to ensure all transactions happened as recorded..

HoughLinesP A function provided by the OpenCV computer vision library that gets the extremes of straight lines it detects. The algorithm works by plotting points as sinusoids and finding the number of intersections. Each intersection then indicates that the points of those sinusoids belong to the same line (OpenCV Team, n.d.).

Reinforcement Learning A set of genetic algorithms that try to find the optimal state-action pairs in order to maximize the outcome of a predefined reward formula.

Unity An open-source game engine used for creating 2D or 3D games..

Chapter 1

Introduction

This introductory chapter provides the reader with background and contextual information regarding this project.

1.1 Background Information

This report is a mandatory deliverable for the Graduation Project (STG2) module offered by Fontys University of Applied Sciences (Fontys), with the deadline being on the 13th of January 2020. It is the final deliverable in combination with the final presentation, and indicates the end of the graduation internship. The report exists to document the assignment, the final status of the project for the time scope of the graduation internship, as well as the process and research done as a form of substantiation.

1.2 Company Description

Seacon's mission is to be the logistics chain director, with a focus on overseas logistics, forwarding and distribution and supply chain solutions. Warehousing forms a basis that is linked to the other core services. Since its foundation in 1985, Seacon has grown to boast more than 700 employees, at various strategic locations throughout Europe.

1.3 Problem Description

Seacon owns more than 235000 m^2 of warehouse capacity, distributed across numerous locations. All these warehouses currently make use of employees that perform inventory control & management tasks. Among these task are checking (specific) locations, counting the amount of items on a pallet, damage checking, and others. All this is currently done by sending employees to one or more locations to the aforementioned tasks manually. This kind of work is tedious and is generally not considered a desirable job by most, and thus employees are getting scarcer. An example of a warehouse layout can be found in figure 1.3, where the rectangle boxes represent racks, and AA/AB/AC represent the unique IDs of each side of a rack.

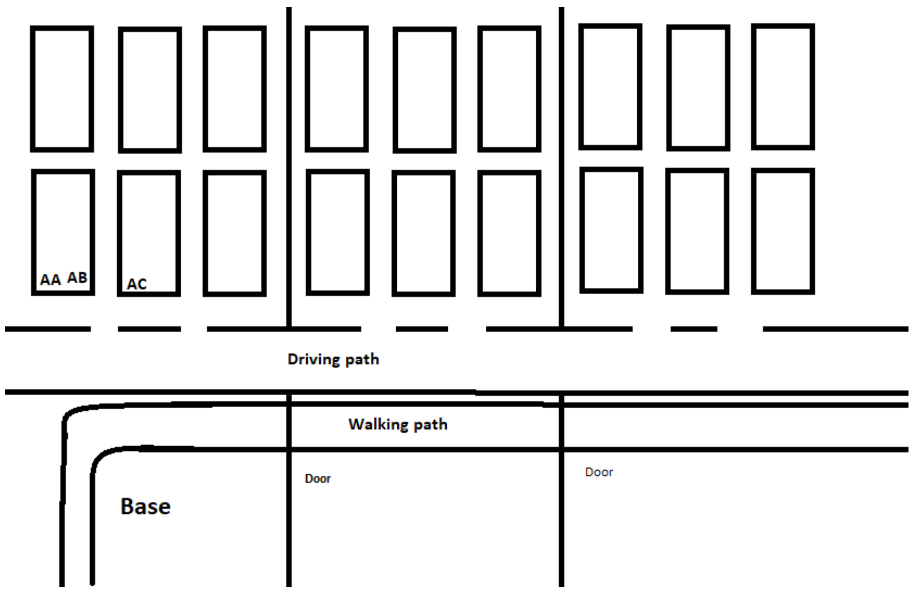


Figure 1.1: Example layout of a warehouse.

1.4 Context

In order to combat employee scarceness, Seacon is looking to develop their own semi-autonomous solution for inventory and control management in their warehouses. All the tasks that the drones will eventually be responsible for are displayed in figure 1.2. Before all those tasks can be executed however, it is essential for the drone to fly a desired path without failure, which the collision avoidance solution will be used for.

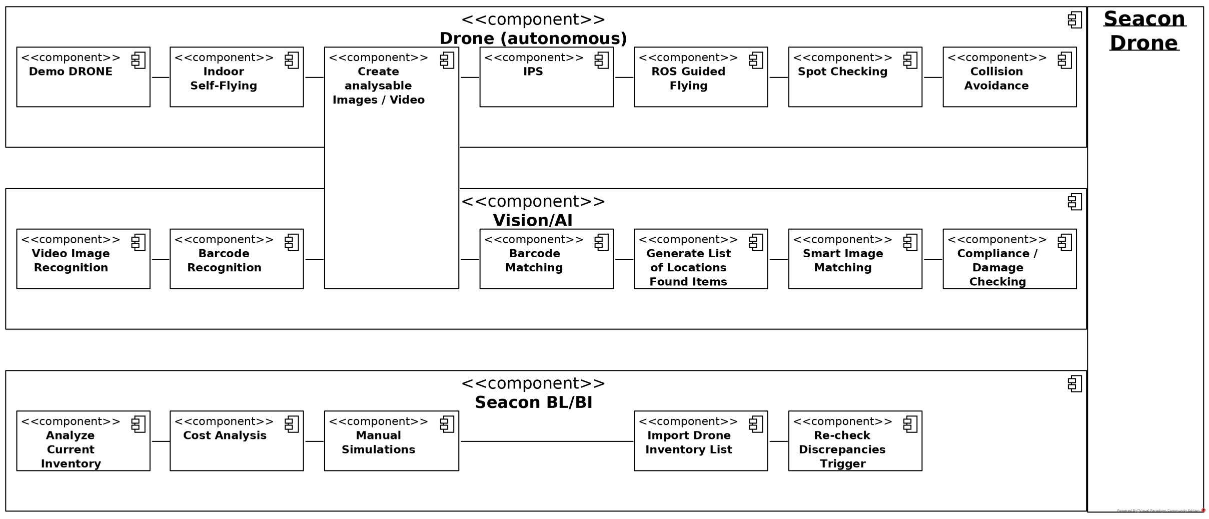

<<component>> Drone (autonomous)
<<component>> Demo DRONE
<<component>> Indoor Self-Flying
<<component>> Create analysable Images / Video
<<component>> IPS
<<component>> ROS Guided Flying
<<component>> Spot Checking
<<component>> Collision Avoidance
Seacon Drone
<<component>> Vision/AI
<<component>> Video Image Recognition
<<component>> Barcode Recognition
<<component>> Barcode Matching
<<component>> Generate List of Locations Found Items
<<component>> Smart Image Matching
<<component>> Compliance / Damage Checking
<<component>> Seacon BL/BI
<<component>> Analyze Current Inventory
<<component>> Cost Analysis
<<component>> Manual Simulations
<<component>> Import Drone Inventory List
<<component>> Re-check Discrepancies Trigger

Figure 1.2: Roadmap of the cycle count drone project.

Due to the image size, the states the drone will traverse through can be found in appendix A. To display how tasks will eventually integrate, the scanning of bar-codes has also been added as an example task. At a later point in time, however, this could be replaced with a generic task state containing all the specific tasks as sub-states.

1.5 Reporting Structure

The remainder of this report will first cover a project description that lists stakeholders and defines the scope and risks. Secondly, possible approaches are described. Thirdly, the first developed concept using reinforcement learning is covered. This also includes information about the development of a simulation environment. Fourthly, the second developed concept using a state machine implementation is described. This chapter contains a requirements analysis, a state machine design, and information about the usage of computer vision for collision avoidance. Fifthly, a summary of the results is given and a conclusion is drawn from that. Finally, recommendations on how to proceed with this project are given.

Chapter 2

Project Description

This chapter provides general information regarding the project, and can be considered an introductory chapter of the analysis done. Included is information about stakeholders, the scope, information about the drone, the risks, and a justification.

2.1 Stakeholders & Affiliates

Within this project, there are 3 groups of stakeholders: The student, Seacon Logistics, and Fontys. The latter 2 groups shall throughout this document be referred to as either "Seacon" or "the company", and "Fontys" or "the university", respectively. The table below lists individual stakeholders, to which group they belong to, and what their roles are.

Name	**Company/Institute**	**Role**	**RACI Role**	**Email**
Tristan van Vegchel	Fontys/Seacon Logistics	Student/Developer	Responsible	Tristan@vanvegchel.eu
Sander Bruinsma	Fontys	Tutor	Informed	S.bruinsma@fontys.nl
Geert Monsieur	Fontys	Examiner	Informed	G.monsieur@fontys.nl
Kai-Arne Reiter	Seacon Logistics	Company Supervisor	Consulted, Informed	Kreiter@seaconlogistics.com
Mark Vromans	Seacon Logistics	Manager Engineering & IT	Accountable, Informed	Mvromans@seaconlogistics.com
Wilfried Beerens	Seacon Logistics	Manager Shared Services Center	Informed	Wbeerens@seaconlogistics.com
Fred Lemmen	Seacon Logistics	Senior Application Specialist	Consulted	flemmen@seaconlogistics.com

Table 2.1: List of stakeholders with their respective roles.

Stakeholders, aside from their regular roles, also have a Responsible-Accountable-Consulted-Informed matrix (RACI matrix) role assigned to them. For a description of the roles, please refer to table 2.2.

Abbreviation letter	Definition	Description
R	Responsible	Who is assigned to work on this task?
A	Accountable	Who has the authority to take decisions and who will be held accountable for the consequences?
C	Consulted	Who are considered stakeholders and who can provide more information about this task?
I	Informed	Whose work depends on this task and who has to kept updated about the progress?

Table 2.2: Descriptions of the RACI matrix roles.

2.2 Minimal Product

The minimal product is considered to be the bare minimal functionality for a project to be considered successful. In the case of this project that is to be able to supply a drone with a solution that lets it move along one side of the rack, while preventing itself from colliding with objects. When talking about objects, the following things are generally meant:

- Pallets
- Racks
- Forklifts
- People
- Drones

As Seacon eventually wants to make use of disposable drones, it is essential for the drone to have as few sensors as possible. Unless it is proven impossible with just the use of a monocular camera-equipped drone, however, the use of additional sensors is advised against.

2.3 Drone Description

This project will make use of the DJI Tello drone (figure 2.3). The drone has been made available by Seacon, and was chosen based on its price, sturdiness, API availability, and functionality. As Seacon wants to make use of disposable drones in the future, the amount of sensors should be kept to a minimum. The Tello drone is equipped with a front-facing color 720p camera, a vision positioning system that makes use of an infrared sensor pointing to the ground below it, and a collision detection system that instantly stops the propellers upon contact (Ryze Robotics, n.d.).

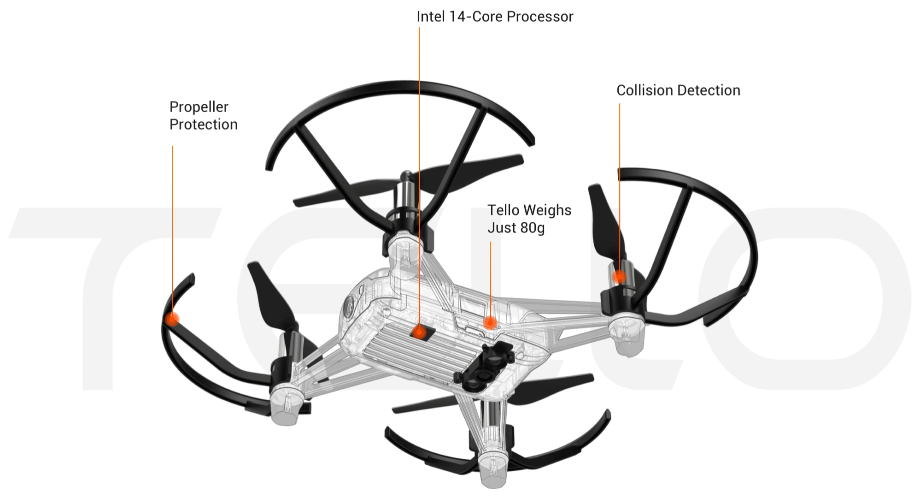


Figure 2.1: Image of the DJI Tello drone. Source: (Ryze Robotics, n.d.)

2.4 Time Scope

This project has been carried out over a period of 5 months, starting from September 2019 until January 2020. Within this period a couple of milestone dates exist, which were defined by Fontys. These dates can be found in the table below:

Deliverable	**Deadline**
Project Plan	27/09/2019
Midterm Report	21/10/2019
Midterm Presentation	11/11/2019
Thesis Report	13/01/2020
Final Presentation	27/01/2020

Table 2.3: List of school deadlines.

2.5 Risk Analysis

In order to minimize damage caused by risks, a risk analysis has been carried out. This has resulted in the creation of a risk matrix, which, due to its size, has been moved to appendix B. The matrix contains a list of risks, the consequences, and its scores based on the likelihood and impact. It also contains a mitigation plan to reduce its score, together with a set of new scores based on the effect of the mitigation.

2.6 Project Justification

Description	**Estimated values**
Amount of locations	80000
Times counted	3
2nd and 3rd counts	1000
Total locations ((amount of locations + 2nd and 3rd counts) * times counted)	243000
Average time (seconds) per location	30
Total work hours per annum	2025
Amount of employees	2
Warehouse employee cost per hour (euros)	20
Total costs warehouse employees (euros)	81000
Total costs (including office employees and forklift costs) (euros)	95000

Table 2.4: Estimated annual costs of cycle counts for 2020, based on a 30 second average per location.

One might wonder why instead of improving inventory control processes not improve picking and storing processes. While it is true that, in the absolute ideal case where storing and picking happens flawlessly, maintenance processes such as cycle counts wouldn't need frequent occurrences. However, considering that humans are involved in these processes it is safe to assume human mistakes will happen. Moreover, as warehouse employees are getting scarcer, the standard for hiring new employees keeps lowering, which results in more mistakes happening.

According to Centraal Bureau voor de Statistiek [Statistics Netherlands] (CBS), the working sector for transport and storage had roughly 8000 work-related injuries with a 4+ days absence occur in 2017. This ranks this sector the 4th highest sector with the most reported 4+ days absence work-related injuries, and the second-highest sector when looking at the percentage of employees sustaining injuries in comparison to the total number of employees in that sector (CBS, 2018).

As seen in table 2.4, the estimated costs for 2020 is roughly €95000, of which €81000 goes to employees performing the cycle counts. Using drones could reduce the number of hours of those employees significantly, and thus reduce the largest cost in the whole cycle count process. It is important to note that the numbers used in this table have been altered slightly for privacy reasons.

Chapter 3

Approaches

This chapter covers the research done on potentially feasible Artificial Intelligence (A.I.)-based approaches.

3.1 Multi-Objective Deep Q Networks with Subsumption Architecture

Originally described by Rodney Brooks, it could be considered the opposition of A.I.. This is due to the fact that subsumption architecture makes use of sensory input to layer competences, instead of being guided by mental/behavioral-based algorithms like A.I. algorithms typically are (Brooks, 1985).

A paper by Tomasz Tajmajer describes a solution that combines a multi-objective Deep Q Network (DQN) with signal suppression known from subsumption architecture to control a floor cleaning robot in a physical environment. As the paper explains, it makes use of separate DQNs (A deep learned form of reinforcement learning) to determine the next action for each of the sub-tasks (floor cleaning, collision avoidance, recharging), which then get combined into one singular output that determines the next action. To create a single output, however, signal suppression is used to create a hierarchy among sub-tasks. For example, if the cleaning robot were to simultaneously have to avoid collision and recharge, it would most likely end up in it colliding as one task commands the robot the move and the other commands it to halt. To avoid this, the collision avoidance task is able to suppress the recharge task in order for the robot to prioritize its safety first (Tajmajer, 2017). This approach could be used for this project, where example sub-tasks could be collision avoidance and moving to a location.

3.2 Object Detection-Based

Single-Shot Detector (SSD) is a method that uses a convolutional neural network to detect objects in a single frame. Its detection speeds are very high, making it very suitable for real-time detection (Liu et al., 2016). An approach could be to train an SSD to detect common objects that the drone needs to avoid. As the sizes of the objects are known roughly, a distance estimation can be made based on the pixel size. Alternatively, a Region-based Convolutional Neural Network (R-CNN) can be used. As the name suggests, it differs from a regular convolutional neural network in the way that it adds another layer to the Faster R-CNN model that creates a segmentation mask around the object in the bounding box, without the significant loss of performance (real-time speed of 5 frames per second) (He et al., 2018). Just like the

Faster-R-CNN it is based on, it makes use of Region Proposal Network that creates rectangles around potential objects and scores them based on their membership to object classes vs. the background (Ren et al., 2016). A project by Waleed Abdulla presents an implementation of a Mask R-CNN (Abdulla, 2017), which could then be used to estimate the distance with. While the distance estimation would be more accurate with the latter approach, it comes at the price of a drop in processing speed and thus frame rate.

3.3 Unity Machine Learning Agents

Unity is an open-source game engine used for creating 2D and 3D games/simulations. In 2017, Unity introduced a toolkit called Machine Learning Agents (ML-Agents), which enables users to use reinforcement learning on their Unity environments (Juliani, 2017; Juliani et al., 2018). The approach would then be to create a somewhat realistic simulation of a warehouse to train an algorithm on, and then later use the trained algorithm on a real drone. An example of this would be the solution presented by Adam Kelly, who created a simulation for training airplanes to fly through waypoints (Kelly, 2019).

An alternative approach could be to make use of imitation learning instead of regular reinforcement learning to train the algorithm. At a Unity event called Unite, Unity gave a number of demonstrations of ML-Agents applications. Among these demonstrations was a racing game, where the algorithm learned to copy the human player's behavior through imitation learning (Juliani and Oster, 2018; Juliani et al., 2018). This could be used to learn the drone to autonomously fly to certain points in the warehouse while avoiding collisions.

Chapter 4

Concept 1: Unity Machine Learning Agents

The first concept chosen was the ML-Agents approach mentioned in section 3.3. This chapter describes the concept of this approach, the reasons for choosing this approach over the others, the simulation built, and the results.

4.1 Concept Description

As the drone only comes with a monocular, front-facing camera to sense its environment with, either the use of traditional computer vision-based or an A.I.-based algorithm (or a combination) seemed like the most straightforward options. A high-paced and somewhat chaotic environment such as a warehouse seemed like an interesting case for A.I.. Seacon was particularly interested in this solution as an A.I. solution might have had the ability to simultaneously and seamlessly handle multiple tasks at once. This resulted in the creation of a concept where drones would start from a base station located somewhere in the warehouse, and would fly to desired locations while avoiding collisions with objects. Since the scope is limited to a single warehouse layout, and warehouses often having random objects/people at unpredictable locations, the simple use of a map was not considered a robust solution. Thus, an Reinforcement Learning (RL) solution was opted for.

4.2 Preliminary Research for Simulation

As drones are relatively expensive, have a limited battery life, and have a risk of getting damaged it was decided to develop a simulation. As the most-widely used language for RL is Python, it was decided to also have a Python-native simulation environment. This would ease exchanging values between the simulation and the RL agent. Multiple existing drone flight simulators were created and published by Simon Levy (Levy, 2019). Out of these projects the PyQuadSim (see figure 4.2) project seemed most suitable, as it was based on Python and lightweight. However, in the end it was decided that to create an own simulation as it requires for it to be as realistic as possible to the real world application (Fridman, 2019). To shorten development times needed, the Unity engine was chosen.

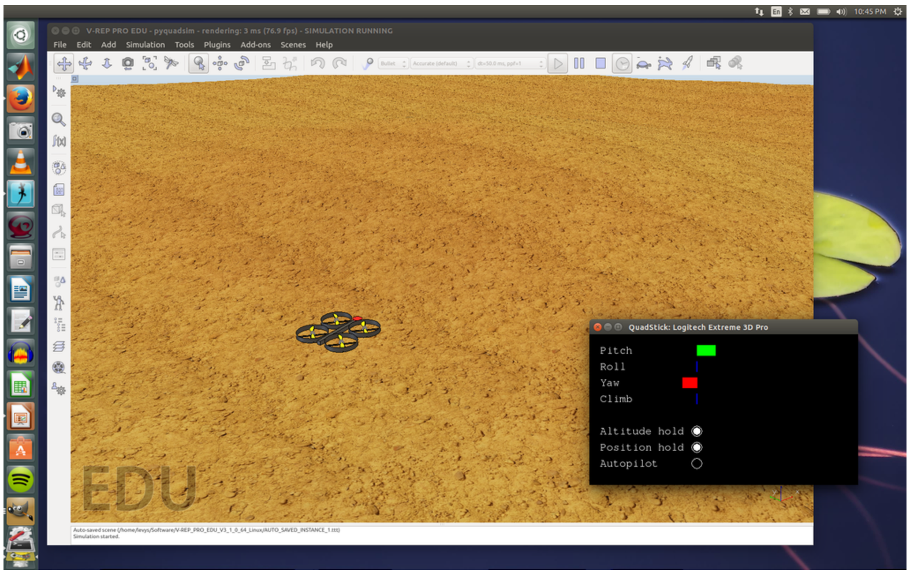

V-REP PRO EDU - pyquadsim - rendering: 3 ms (76.9 fps) - SIMULATION RUNNING
File Edit Add Simulation Tools Plugins Add-ons Scenes Help
QuadStick: Logitech Extreme 3D Pro
Pitch
Roll
Yaw
Climb
Altitude hold
Position hold
Autopilot
EDU

Figure 4.1: Screenshot of the PyQuadSim project. Source: (Levy, 2014)

.

Unity was chosen over alternatives for 2 reasons. Firstly, the company had experience with developing products using this engine. This opened up the opportunity for the company to do code reviews and generally being able to provide tips. Moreover, a realistic model for the drone and a variety of warehouse assets were available. The second and most important reason for choosing Unity, however, was ML-Agents.

4.3 Unity Simulation

The original simulation can be seen in figure 4.3. It was made so each time the warehouse was generated, it would have a random size and amount of racks ranging from 1 to 8 racks. It was also equipped with an option to turn on or turn off training mode. Turning on training mode would minimize graphical load by replacing the models with simpler shapes and textures, as seen in the right image of figure 4.3.

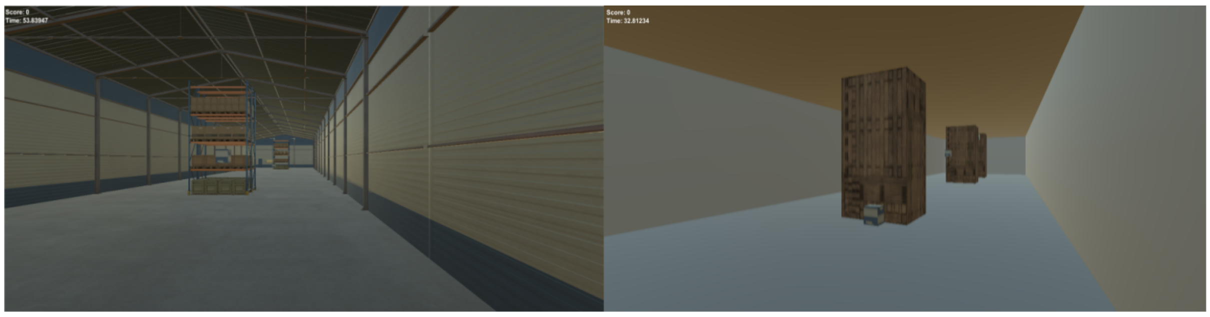


Figure 4.2: First design of the Unity warehouse simulation.

Based on feedback by the company the design was changed to feature 2 long rows of racks instead, as this was considered closer to the real situation. This was changed as displayed in figure 4.3, while maintaining the option for toggling training mode and a variable warehouse length. Both designs also featured targets spawning in at random positions in front of the racks. These targets functioned as an indication for where the drone should fly, and would grant the drone an increase in score whenever the drone flew over it. The simulation was made so that the view on screen simulated the view a real drone would have through its front-facing camera.

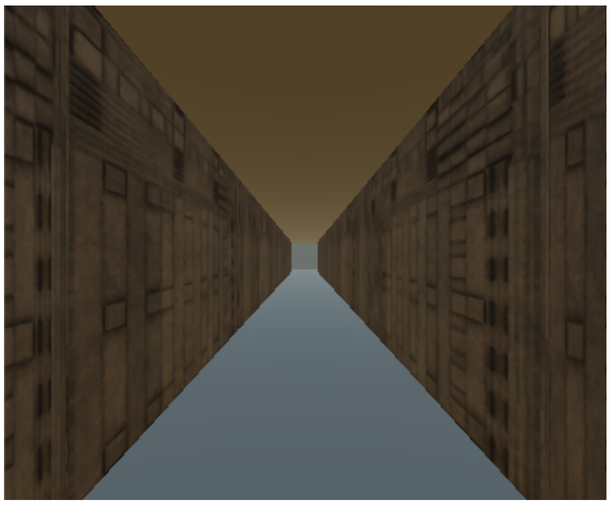


Figure 4.3: Updated design of the simulation in training mode.

4.4 ML-Agents Classes

The implementation of the simulation with ML-Agents contains 4 major parts: generator classes, the Unity package, the ML-Agents package and its inheriting classes, and the class for reading a configuration file. Since the Unity package is the default package that all c# Unity projects use, it will not be discussed here. A class diagram can be found in appendix C.

Generator Generator is an abstract class that has concrete implementations for spawning the drone, warehouse, and the targets. Using the strategy design pattern, all the generators are registered and invoked by the DroneAcademy class.

DroneAcademy DroneAcademy inherits from the Academy class within the ML-Agents package. It is responsible for all operations around the environment during training, for example destroying or adding new objects to the scene.

DroneAgent DroneAgent inherits from the Agent class within the ML-Agents package. The object containing an implementation of the Agent class is meant to be controlled by a neural network (or "brain" in the context of Unity) during training.

ConfigHandler ConfigHandler reads an external configuration file that contains information about how to generate the scene. Examples include the size of the warehouse and whether or not to enable training mode.

4.5 Reinforcement Learning & Imitation Learning

For the RL implementation the standard models provided by ML-Agents were used. The RL implementation was given the drone's coordinates, horizontal and vertical velocity, camera input, as well as the target's coordinates as input. The drone (or "agent" in RL lingo) then was given 10-60 seconds (based on the warehouse size) to reach its target, and was rewarded based on its time left after reaching its goal. The agent was also mildly penalized for moving, in order to motivate it to keep its movement to a minimum. Once the agent had reached its target, the environment would reset and the warehouse would be regenerated using a random size, and the end of a so-called episode would then be reached. The ending of an episode means that the total reward score for that lifetime is finalized. In case the drone would crash into something, the episode would end immediately and the total reward score would be the lowest possible. Unfortunately, after having run several training sessions spanning somewhere between 10 minutes and 3 hours, the agent did not seem to learn. It kept flying a similar pattern, where it would fly up and slightly left or right, crashing into the ceiling.

As alternative, another model provided by ML-Agents was used. This model, however, was for Imitation Learning (IL). While they are both genetic algorithms, the major difference between IL and RL is that IL does not make use of a predefined reward function. Instead, it tries to derive that function by imitating an expert (human) player. As a result, the simulation was slightly altered to spawn 2 identical warehouses that allowed for multiplayer (human vs. agent), as seen in figure 4.5. To monitor progress, the simulation was altered to display both the view of the human-controlled drone and the agent-controlled drone in a split screen manner. This, however, also did not show any significant results. Even after having provided demonstrations for an hour, the agent kept moving slowly until it crashed in its nearest object, and did thus not show any sign of learning.

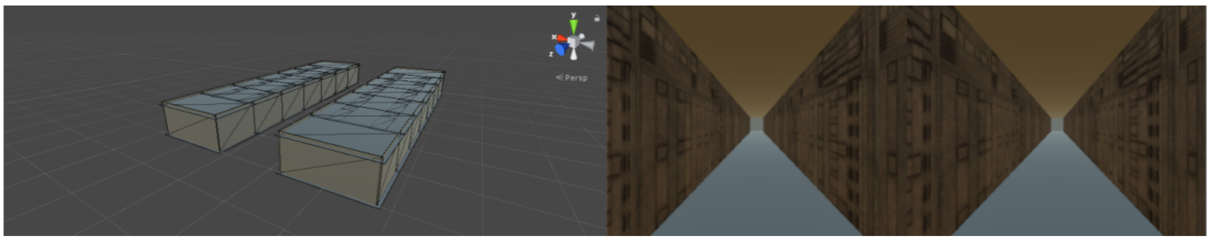


Figure 4.4: Simulation altered for running for imitation learning. The left image shows the warehouses from the outside, and the right image shows the view seen during training.

Chapter 5

Concept 2: State Machine Implementation

Based on feedback received during the midterm presentation, it became clear that drastic changes needed to happen, mainly due to an improper requirements analysis. Since the concept described in chapter 4 did not show promising results, it was decided to switch to a new concept in order to reduce the risk of not having a (partially) working product at the end. Through a brainstorm session with the company a new concept was developed. In order to combat time constraints, it was decided to limit the scope to just one side of a rack, instead of covering the whole warehouse. Further details regarding this concept are explained in the remainder of this chapter. Installation instructions and a guide for running the project can be found in appendix D.

5.1 Product Functions

The goal of this project is mainly focused on developing a solution for providing a drone with collision avoidance. However, along the course of the project it became evidently clear that a indoor self-flying (figure 1.2) component was also necessary. From here on out, the functions could be divided among the 3 dimensions the drone has to interact with as follows:

- X-axis:
  - Move the drone sideways
  - Stop the drone at the end of a bar
  - Detect objects in the drone's trajectory
  - Avoid collisions with objects in the drone's trajectory
- Y-axis:
  - Keep the bar in the center of the drone's camera
  - Change the drone's altitude to switch layers
  - Keep track of the current layer and the highest layer
- Z-axis:
  - Estimate the pixel size of the bar
  - Keep the drone to maintain distance from the bar

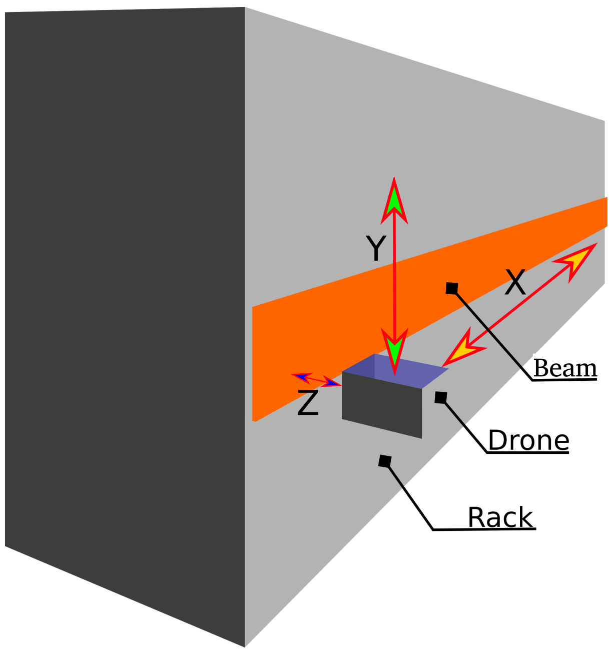


Figure 5.1: Visual representation of the single rack concept.

This list is originally taken from the Software Requirements Specification (SRS) in appendix G. Figure 5.1 contains a visual representation of the concept with its functions.

5.2 State Machine

The initial state design featured a concept where the drone correct its positions for safety, move sideways viewing the beam at set intervals and then do a check for objects in its trajectory. This design is displayed in appendix E. However, it was decided to shift to a design that selects a state based on a set of criteria (see table 5.1) so that states such as checking for collision or locking-on can happen more frequently. For the second state machine design, please refer to appendix F. It is important to note that as of writing this document, the states "moving layer" and "returning home" have not been implemented yet due to time constraints. The state "avoiding collision" has only been partially implemented, where it only determines if an object is too close and then proceeds to land accordingly. Descriptions of all implemented states can be found in table 5.2.

			Transition to state:
Criteria:	Locked on?	Distance interval reached?	
	False	False	Locking on
	False	True	Locking on
	True	False	Advancing
	True	True	Rotating to hall

Table 5.1: Table displaying state transitions from the facing bar state, based on 2 criteria.

For simplification, the currently implemented state machine can be grouped into 3 group states: locking on, checking for collision, and advancing. Here, locking on refers to centering the beam in its view and maintaining an appropriate distance, checking for collision refers to rotating the drone to check the hallway for objects in its trajectory, and advancing refers to moving the drone sideways along the beam. The criteria "Locked on?" is currently set to false after having completed any of the aforementioned group states (aside from "locking on", which sets it to true again). "Distance interval reached?" is set to true after exiting the advancing group state, and back to false after exiting "checking for collision".

State	**Description**
Facing beam	The drone is hovering in front of the beam while facing it. Also acts as a central state.
Locking on	The drone is positioning itself to center the beam and maintain its distance from it.
Advancing	The drone is moving sideways along the beam.
Rotating hall	The drone is rotating towards to hall to prepare to check for collisions.
Detecting collision	The drone is using background subtraction to check for objects in its near trajectory.
Avoiding collision	The drone determines if an object is too close and lands to avoid collision if necessary.
Rotating beam	The drone rotates back so it is once again facing the beam.

Table 5.2: Implemented states with descriptions of what happens during their executions.

5.3 Implementation

The implementation is written in Python and makes use of 3 main components: OpenCV, transitions, and the Tello-openpose project. OpenCV is a computer vision library (OpenCV Team, 2019), transitions is a Python package that provides state machine implementation functionality (Yarkoni, 2019), and Tello-openpose is a project by Github user Geaxgx that enables a Tello drone to track a human and follow commands via gestures (Geaxgx, 2019). The last component mentioned was used as the basis for this project, but has since been modified significantly.

The product contains the following scripts with the following responsibilities:

Main.py: Used to run the product, primarily responsible for invoking the other scripts, connecting to the drone, and running a loop to obtain the frames.

DroneController.py: Communication script between the other scripts and the drone. It stores information about the drone, handles the logs, and provides functionality for controlling the drone either through functions or by keyboard. This script was based on the Tello-openpose project, and makes use of a Python package called Tellopy that provides a set of functions for the scripted controlling of a Tello drone (Hanyazou, 2018).

BeamDetector.py: Provides functions that get an estimate of the edges of a beam. This happens by using OpenCV to get a binary (either black or white pixels) image, get the contours, and then obtain all straight lines from the contours using a function called HoughLinesP. To filter out noise, the script also gets the average Y-coordinate of all lines to create a pivot, and then groups them together into a lower-half and upper-half group. The respective averages of those 2 groups then form the upper-edge and lower-edge of the beam.

DistanceEstimator.py: Provides functions for getting the distance between the center of the beam and the camera center, as well as a function for estimating the distance based on the size of the beam in pixels. The former is done by providing the frame width, frame height, and the minimal and maximal Y-coordinate found using the functions of BeamDetector.py.

CollisionDetector.py: Provides functions that can distinguish foreground objects from background objects, get the contour size from the foreground objects and use that to very roughly estimate the proximity based on the amount of space the contour takes up in the frame. For reference, please look at figure 5.3. The top two images are frames directly from the drone camera, and the bottom two are the same frames where background subtraction has been applied. In the images of the left column the drone is at a safe distance, while in the right column it is too close, hence why the bottom right image has a ”Danger” tag in it.

StateMachine.py: Makes use of the transitions Python package to define the states, the possible state transitions, and the conditions for transitioning. Also stores a time stamp of the time when the most recent state enter has occurred.

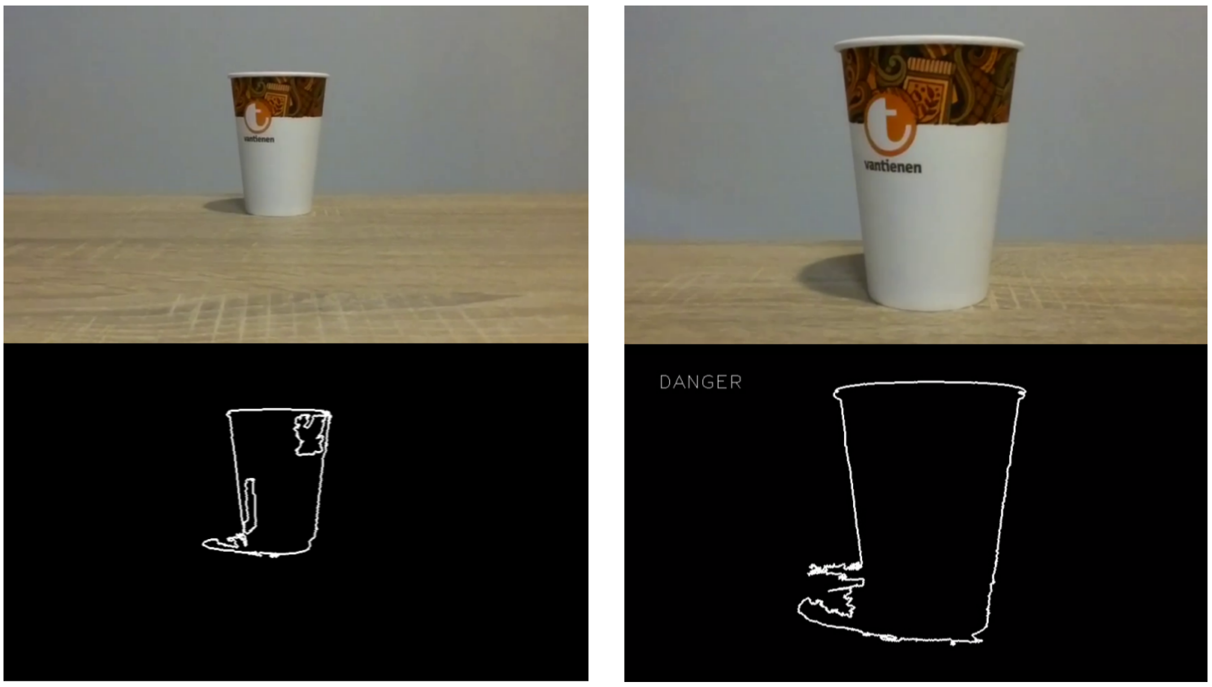


Figure 5.2: Set of images demonstrating collision detection.

StateMachineActions.py: Acts as a bridge between StateMachine.py and DroneController.py. It has an execute function that checks the state of the StateMachine instance and invokes a function accordingly. It also checks when a state can be exited, and updates the criteria seen in table 5.1. For example, when the StateMachine instance has entered the ”advancing” state, StateMachineActions will invoke the function in DroneController that will make the drone move sideways until the interval is reached. Then it will update the criteria and use the transition to go back to the ”facing bar” state.

FPS.py & HUD.py: As Main.py displays the camera frames in a window, HUD.py adds and updates textual information to each frame. FPS.py calculates the frames per second, which is in turn used by HUD.py. The code of both these scripts originated from the Tello-openpose project.

5.4 Limitations & Problems

The current product faces 2 major issues. The biggest issue is the limitations of classical computer vision algorithms, on which the product is highly dependent on. A lot of averaging has been done in an attempt to minimize noise, however it seems that the distance and lighting range in which the product operates normally is still rather small. Another major issue is the communication with the current drone. It only accepts one command at a time sent by an UDP-protocol ("single shot" message-oriented communication protocol), which might cause the software and the drone to desynchronize. To clarify, listing 5.1 displays the debug log while running the Main.py script. Each 0.2 seconds the current state is printed. However, while in line 2 and 3 it shows the command is received by the drone fairly quickly after switching states (at least within 0.2 seconds), it does not to seem to be the case when rotating back at lines 6-8. This causes the drone to not rotate back all the way (due to the duration of the rotation being an estimate based on speed and distance) and ultimately causes it to stop functioning properly.

```
 1  facing bar
 2  Rotating to hall
 3  Tello: 14:44:45.588:  Info: clockwise(val=100)
 4  ...
 5  Avoiding collision
 6  Rotating to bar
 7  Rotating to bar
 8  Tello: 14:44:48.224:  Info: clockwise(val=-100)
 9  Rotating to bar
10  Stopping rotation
11  Tello: 14:44:48.494:  Info: clockwise(val=0)
12  facing bar
```

Listing 5.1: Fragment excerpted from the debug log of the product.

One minor improvement point is that the distance between the camera center and the center of the beam is still measured in pixels, while the distance between the drone and the beam is in metric units. For consistency purposes, this should be changed to the same format.

There are also a couple of limitations caused by the Tello drone. To be able to stabilize itself, it makes use of an infrared camera that points to the surface it is hovering over. However, the requirements for this to work are rather strict. It seems to respond badly to uneven surfaces (think of flying over small objects such as doorsteps), reflective surfaces, and dark or monochrome surfaces. The issue with this is that, especially in the case of uneven surfaces, the drone will force itself to correct its position, and thus overrule any other input. This might cause issues when a drone were to fly over a box on the ground at a low altitude as the script receives very little information about the drone's status, causing it to ultimately desynchronize. Another limitation is its battery life, which, after testing, spans roughly 10 minutes.

Chapter 6

Conclusion

The Warehouse Drone Collision Avoidance project resulted in the development of 2 concepts: an A.I.-based one using Unity's ML-Agents and a state machine implementation one. This chapter gives a short summary of each of the concepts and draws a conclusion.

6.1 Machine Learning Agents & Research

For the ML-Agents concept a Unity environment was developed that could simulate a warehouse with variations. Using the ML-Agents toolkit several training sessions were run. This concept, however, did not show any signs of learning after having trained it for 3 hours for RL and 1 hour for IL. After having done more research, it can be concluded that reinforcement learning as stand-alone solution will not suffice. This is due to the fact that rewards are very sparsely distributed, which is generally handled badly by RL algorithms. Moreover, as environments get increasingly complex, the risk of the algorithm getting stuck in local optima or finding an out-of-the-box answer that gives a better reward than the intended answer increases (Irpan, 2018). An example is the paper by A.I. research company OpenAI on a game where 4 agents are tasked to play hide and seek in teams of 2. In the paper, it is described that eventually the agents started finding several exploits in the physics engine of the game, with an example being abusing the contact physics to force a ramp to go through the wall separating the play area (Baker et al., 2019).

Based on the results and the risk of not having selected a viable approach, it was decided to switch to the state machine concept after the midterm presentation. This was done as a (partially) working product was favored over continuing the research.

6.2 State Machine Concept

Unlike the ML-Agents approach, the state machine concept limited its scope to just one side of a rack. Through the use of a state machine, the product is able to switch tasks based on a set of criteria. Currently included tasks are: finding the vertical distance between the center of the camera and the center of the beam, estimating the distance between the drone and the beam, moving sideways, rotate, distinguish the foreground from the background, and make a rough estimation of how close an object is. Although the product contains an implementation for all these tasks, it is concluded that it is not ready for a practical environment yet. Tests using the Tello drone show that the product still has insufficient control over the drone, and is still too susceptible to noise. Furthermore, the drone still contains some bugs and improvement points, such as the rotation bug mentioned in section 5.4 and the fact that the collision detector its performance is based on well the drone can maintain its position. With some optimization, however, it is believed that this concept could end up being usable in a practical environment such as a warehouse.

Chapter 7

Recommendations

There are many ways to proceed with this project. This chapter describes the recommended ways to proceed, based on the work and research done. The recommendations listed focus on the improvement of the existing implementation, rather than extending it.

7.1 Computer Vision Enhancements

The state machine concept is mostly reliant on its accuracy of its computer vision tasks. Enhancing these should result in a drastic increase in stability. There are, however, many ways to go about this. Currently, all computer vision tasks make use of a single frame. Adjusting this to make use of an average of a small number of frames should help out filtering noise without a significant loss in performance speed.

7.2 Movement Enhancement with PID controllers

One specific improvement possible is the improvement of the beam centering and distance maintenance (locking on state) task through the implementation of Proportional-Integral-Derivative (PID) controllers. PID controllers makes use of a feedback loop control mechanism to regulate values. Using the beam centering as example, a PID controller would make use of the present difference between center points (Proportional), how far it has already moved in previous iterations of the loop (Integral), as well as an estimate of how far it will end up afterwards (Derivative) for determining how much it should move the drone up or down. Proper use of this should result in not only in increase in positioning accuracy, but also an increase in performance speed.

The Tello-openpose project that this product was based on made use of such controllers for tracking a human (Geaxgx, 2019). Due to difficulties with tuning them and time constraints, however, it was decided to leave them out at the time.

7.3 Object Segmentation

Continuing the research, the computer vision part can also be enhanced using the object detection/segmentation approach described in section 3.2. Using deep learned mask segmentation on the beam could provide a more accurate and robust solution to the distance estimation and beam centering tasks. It could also be used for collision detection, similarly to how the current implementation works. The model could be trained on common objects it has to mask. A distance estimation can then be made using its known average size together with the size of the mask.

References

Abdulla, W. (2017), 'Mask r-cnn for object detection and instance segmentation on keras and tensorflow'.
URL: *github.com/matterport/Mask_RCNN*

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B. and Mordatch, I. (2019), 'Emergent tool use from multi-agent interaction'.
URL: *openai.com/blog/emergent-tool-use/*

Brooks, R. (1985), 'A robust layered control system for a mobile robot'.
URL: *apps.dtic.mil/dtic/tr/fulltext/u2/a160833.pdf*

CBS (2018), 'Meer arbeidsongevallen met vier of meer dagen verzuim'.
URL: *www.cbs.nl/nl-nl/nieuws/2018/29/meer-arbeidsongevallen-met-vier-of-meer-dagen-verzuim*

Fridman, L. (2019), 'Mit 6.s091: Introduction to deep reinforcement learning (deep rl)'.
URL: *www.youtube.com/watch?v=zR11FLZ-O9M&t=845s*

Geaxgx (2019), 'Tello-openpose'.
URL: *github.com/geaxgx/tello-openpose*

Hanyazou (2018), 'Tellopy'.
URL: *github.com/hanyazou/TelloPy*

He, K., Gkioxari, G., Dollár, P. and Girshick, R. (2018), 'Mask r-cnn'.
URL: *arxiv.org/pdf/1703.06870.pdf*

Irpan, A. (2018), 'Deep reinforcement learning doesn't work yet'.
URL: *www.alexirpan.com/2018/02/14/rl-hard.html*

Juliani, A. (2017), 'Introducing: Unity machine learning agents toolkit'.
URL: *blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents*

Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M. and Lange, D. (2018), 'Unity: A general platform for intelligent agents'.
URL: *arxiv.org/pdf/1809.02627.pdf*

Juliani, A. and Oster, V. (2018), 'Democratize machine learning: Ml-agents explained - unite la'.
URL: *youtu.be/OTMlElDuN3k?t=645*

Kelly, A. (2019), 'Ai flight with unity ml-agents'.
URL: *www.youtube.com/watch?v=vkFhp30XzGk*

Levy, S. (2014), ‘Pyquadsim: A python quadrotor simulator’.
URL: *www.youtube.com/watch?v=oQejljPA6ys*

Levy, S. (2019), ‘Github simondlevy’.
URL: *github.com/simondlevy*

Lexico (n.d.), ‘Definition of ”artificial intelligence”’.
URL: *www.lexico.com/definition/artificial_intelligence*

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A. (2016), ‘Ssd: Single shot multibox detector’.
URL: *arxiv.org/pdf/1512.02325.pdf*

OpenCV Team (2019), ‘About opencv’.
URL: *opencv.org/about/*

OpenCV Team (n.d.), ‘Hough line transform’.
URL: *docs.opencv.org/3.4.9/d9/db0/tutorial_hough_lines.html*

Ren, S., He, K., Girshick, R. and Sun, J. (2016), ‘Faster r-cnn: Towards real-time object detection with region proposal networks’.
URL: *arxiv.org/pdf/1506.01497.pdf*

Ryze Robotics (n.d.), ‘Tello’.
URL: *www.ryzerobotics.com/tello?from=store-product-page*

Tajmajer, T. (2017), ‘Multi-objective deep q-learning with subsumption architecture’.
URL: *arxiv.org/pdf/1704.06676v1.pdf*

Yarkoni, T. (2019), ‘Transitions’.
URL: *github.com/pytransitions/transitions*

Appendix A

Drone States

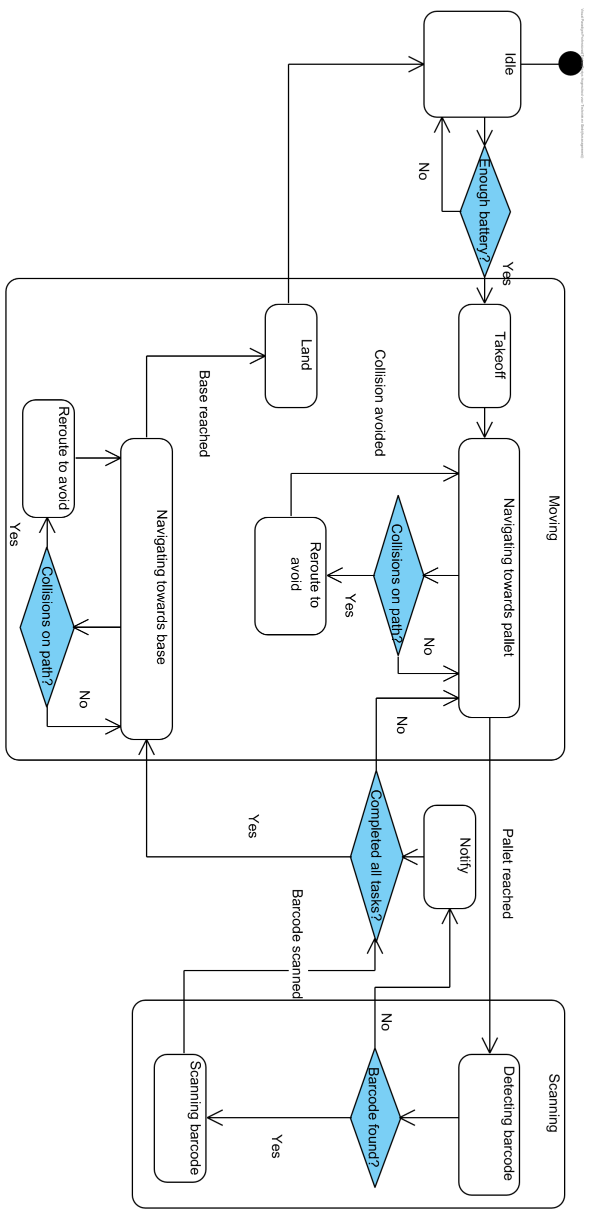
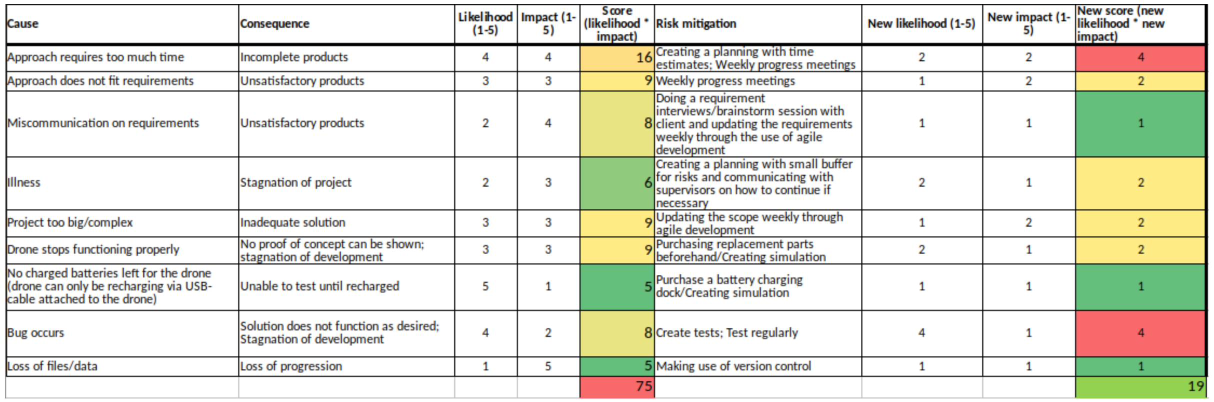


Figure A.1: State model of a drone performing inventory control

Appendix B

Risk Management



Cause	Consequence	Likelihood (1-5)	Impact (1-5)	Score (likelihood * impact)	Risk mitigation	New likelihood (1-5)	New impact (1-5)	New score (new likelihood * new impact)
Approach requires too much time	Incomplete products	4	4	16	Creating a planning with time estimates; Weekly progress meetings	2	2	4
Approach does not fit requirements	Unsatisfactory products	3	3	9	Weekly progress meetings	1	2	2
Miscommunication on requirements	Unsatisfactory products	2	4	8	Doing a requirement interviews/brainstorm session with client and updating the requirements weekly through the use of agile development	1	1	1
Illness	Stagnation of project	2	3	6	Creating a planning with small buffer for risks and communicating with supervisors on how to continue if necessary	2	1	2
Project too big/complex	Inadequate solution	3	3	9	Updating the scope weekly through agile development	1	2	2
Drone stops functioning properly	No proof of concept can be shown; stagnation of development	3	3	9	Purchasing replacement parts beforehand/Creating simulation	2	1	2
No charged batteries left for the drone (drone can only be recharging via USB-cable attached to the drone)	Unable to test until recharged	5	1	5	Purchase a battery charging dock/Creating simulation	1	1	1
Bug occurs	Solution does not function as desired; Stagnation of development	4	2	8	Create tests; Test regularly	4	1	4
Loss of files/data	Loss of progression	1	5	5	Making use of version control	1	1	1
				75				19

Figure B.1: Risk matrix containing the risks with their mitigation plans.

Appendix C

Simulation Class Diagram

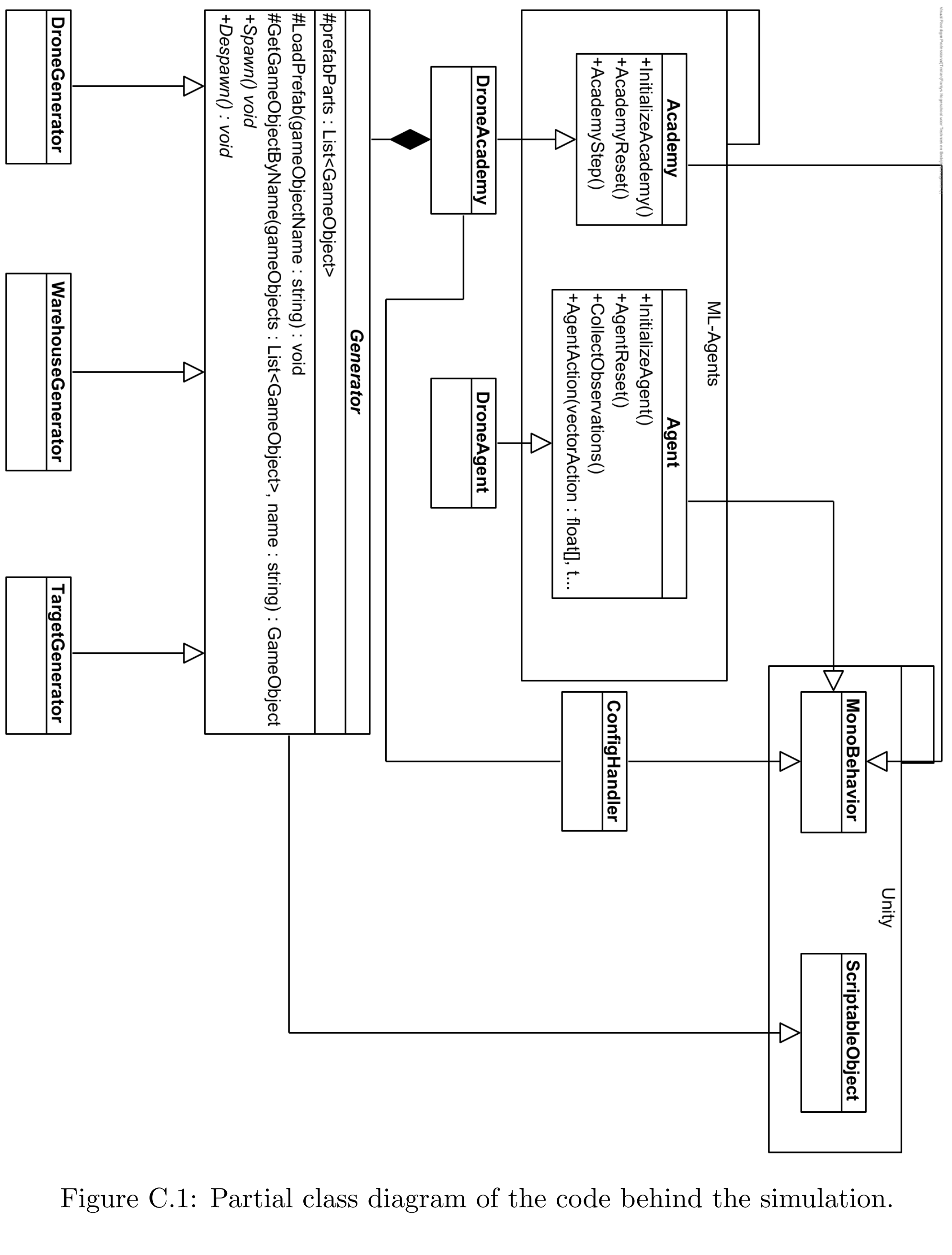


Figure C.1: Partial class diagram of the code behind the simulation.

Appendix D

Setup State Machine Concept

This chapter provides a set of instructions to install and set up the state machine concept. The link to the Git repository has been left out as it private. The project was developed using Python 3.6.9, which is also needed to run the project.

1. Clone the repository to a local machine
2. Switch to branch "heuristic"
3. The project resides in the ./Python
4. Turn on the Tello drone
5. Once it starts flashing a green light, connect to the drone's WI-FI network.
6. Run the file Main.py using the command "python Main.py" in the command line.
7. Press the tab key to launch the drone, and press the "M" key to start the state machine. Note: make sure it is facing the beam before pressing "M".
8. To stop the state machine, press "M" again.
9. To land the drone and kill the program, press ESC. To just land the drone, press backspace.
10. Other controls for manually controlling the drone can be found in the DroneController.py file.

This project is dependent on the following modules which can be installed through pip:

- opencv-python (version 4.1.1.26)
- av (version 6.2.0)
- tellopy (version 0.6.0)
- simple_pid (version 0.2.4)
- pynput (version 1.4.5)

Appendix E

First State Machine Design for Concept 2

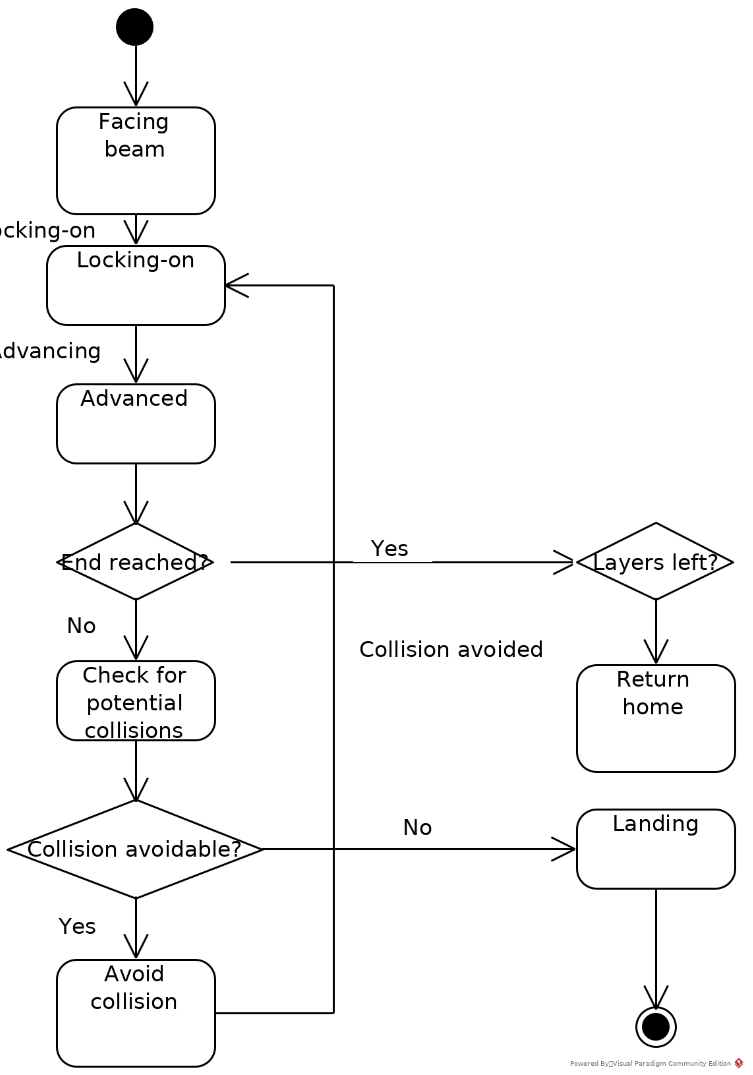


Figure E.1: First state diagram design for state machine concept.

Appendix F

Second State Machine Design for Concept 2

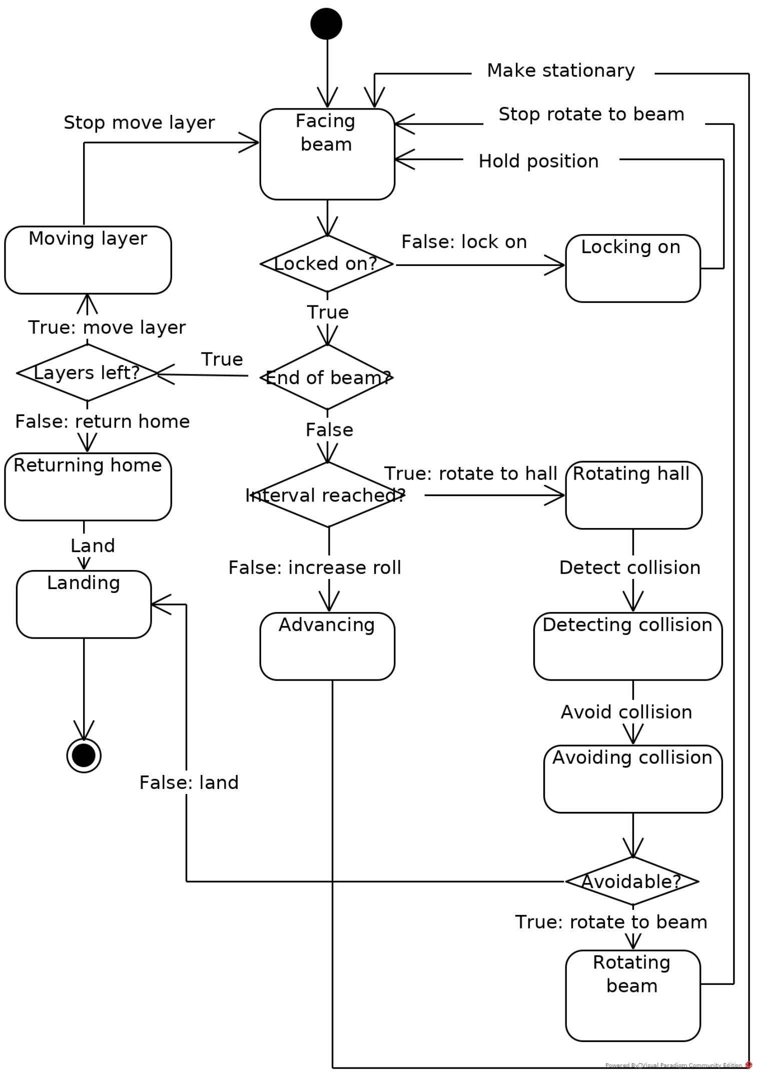


Figure F.1: Second state diagram design for state machine concept.

Appendix G

Software Requirements Specification

Page left intentionally blank to act as cover page.

Software Requirements Specification for Warehouse Drone Collision Avoidance Project

Prepared by Tristan van Vegchel
Seacon Logistics
Approved from version 1.0 onwards

Venlo, created on November 11, 2019

Contents

List of Figures

Revision History

Revision	Date	Author(s)	Description
0.1	11-11-2019	Tristan van Vegchel	Created skeleton
0.1	12-11-2019	Tristan van Vegchel	Created shortened version
0.2	27-11-2019	Tristan van Vegchel	Added introduction
0.3	06-12-2019	Tristan van Vegchel	Added overall description
0.4	23-12-2019	Tristan van Vegchel	Added functional requirements
0.5	01-01-2020	Tristan van Vegchel	Completed first draft
1.0	10-01-2020	Tristan van Vegchel	Document reviewed and approved

List of Abbreviations

A.I.	Artificial Intelligence.
Fontys	Fontys University of Applied Sciences.
STG2	Graduation Project.

Definitions

Action task A task that is directly related to the movement of the drone.

Customer The person, or persons, who pay for the product and usually (but not necessarily) decide the requirements. In the context of this recommended practice the customer and the supplier may be members of the same organization.

Layer A rack in a warehouse allows for multiple pallets to be stacked on top of each other, separated by horizontal beams. A layer consists out of a single beam with the items placed directly on top of it.

Maintenance task A supportive task that allows the drone to be able to perform its original tasks without interference. An example being readjusting the distance from the beam caused by drift.

Might Features that might perform a certain task imply that said feature is generally desired by the customer, but is not listed as a requirement.

Opencv A computer vision library originally written in C.

Product The product is the name used to refer to the product of this iteration of the project, or the product of the warehouse drone collision avoidance project.

Seacon Drone Project The Seacon Drone Project is the name used to refer to the entire project to semi-automate inventory control processes using drones. As of writing this document this includes functionality for self-navigating through the warehouse, collision avoidance, performing cycle counts, and checking specific locations.

Shall Features that shall perform a certain task imply that said feature is both required and essential for the completion of the product. These features are thus considered high-priority features.

Should Features that should perform a certain task imply that said feature is required by the customer, but is given lower priority and is not considered essential.

Status task A task concerned with (partially) retrieving the status of the drone and/or its environment. An example being tracking the vertical center of the beam.

Supplier The person, or persons, who produce a product for a customer. In the context of this recommended practice, the customer and the supplier may be members of the same organization.

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to provide a detailed description of the software product of the Drone Warehouse Collision Avoidance project. It contains the purpose and features of the software, the interfaces, and the constraints under which it must operate. This document is primarily intended for the department of Engineering & IT of Seacon Logistics, as this department will contain/contains the future developers of the next iteration of this product. As this project is developed as a bachelor thesis project, the by Fontys University of Applied Sciences appointed tutors, lecturers, and experts are also considered an intended audience.

1.2 Document Conventions & Standards

This document was created based on the IEEE Recommended Practice for Software Requirement Specifications standard 830-1998.

1.3 Scope

The Warehouse Drone Collision Avoidance system (which is in this document generally referred to as "product") will primarily be responsible for providing a drone with a prototype solution to move along all layers of a single side of a rack in a warehouse, while avoiding collisions.

1.4 Overview

The remainder of this document will feature an overall description of the product and the specific requirements. The former includes the perspective, information regarding a set of used interfaces, the product functions, and information regarding the product its users. The latter contains the specific requirements listed with unique identifiers, and a list of assumptions made in order for the product to function.

Chapter 2

Overall Description

2.1 Product Perspective

The product will be developed as a first step of a larger project, which ultimately has the goal to semi-automate the cycle count processes of the inventory control & management department of the customer's warehouses. The concept was initially developed in order to combat the increasing scarceness of warehouse cycle count employees. An overview of the larger project can be found in figure 2.1.

System Interfaces As the product for which this document is written will be the first component of the Seacon Drone project, the interfaces of the Seacon Drone project which the product will communicate with have yet to be defined.

Hardware Interfaces The product will support Dji Tello drones (Ryze Robotics, 2018). It will developed to be used with a Wi-Fi enabled laptop. The product will provide a set of drone control command keys, which will only work as intended on an QWERTY or AZERTY keyboard layout.

Software Interfaces The product will be developed in Ubuntu 18.04. This will thus be the only operating system and version which will have guaranteed support. Furthermore, the device running the product is required to have a python interpreter version 3.6.x with pip, as this will be the language and package management system for development, respectively. The python-pip packages the software will make use of are:

- opencv-python (version 4.1.1.26) (OpenCV Team, 2019)
- av (version 6.2.0) (Boers, 2019)
- tellopy (version 0.6.0) (Hanyazou, 2018)
- simple_pid (version 0.2.4) (Lundberg, 2019)
- pynput (version 1.4.5) (moses palmer, 2019)

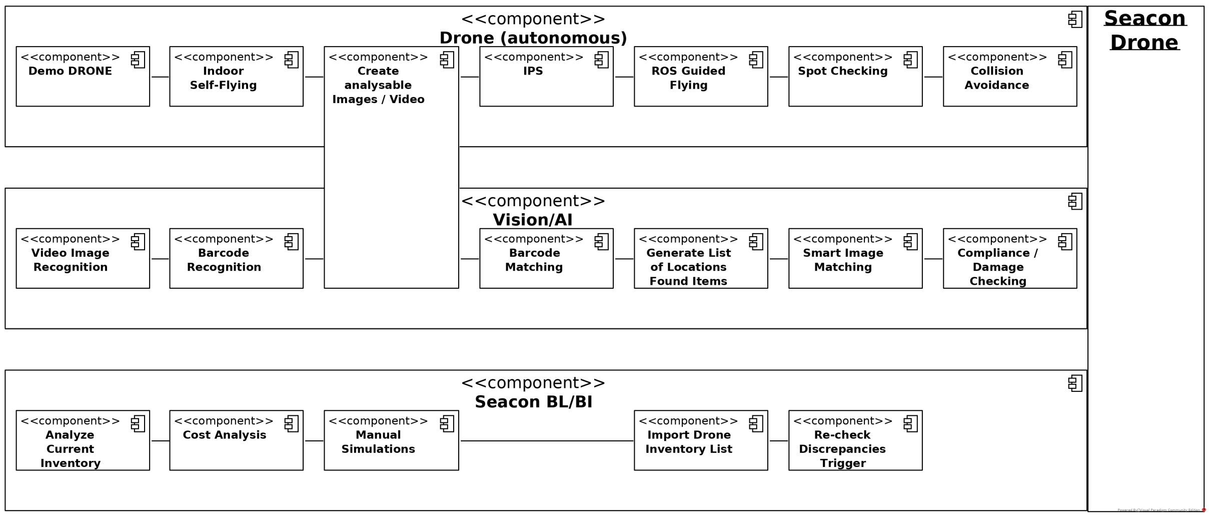

Seacon Drone
<<component>> Drone (autonomous)
<<component>> Demo DRONE
<<component>> Indoor Self-Flying
<<component>> Create analysable Images / Video
<<component>> IPS
<<component>> ROS Guided Flying
<<component>> Spot Checking
<<component>> Collision Avoidance
<<component>> Vision/AI
<<component>> Video Image Recognition
<<component>> Barcode Recognition
<<component>> Barcode Matching
<<component>> Generate List of Locations Found Items
<<component>> Smart Image Matching
<<component>> Compliance / Damage Checking
<<component>> Seacon BL/BI
<<component>> Analyze Current Inventory
<<component>> Cost Analysis
<<component>> Manual Simulations
<<component>> Import Drone Inventory List
<<component>> Re-check Discrepancies Trigger

Figure 2.1: Roadmap of the Seacon Drone Project

2.2 Product Functions

The product can be divided into 3 parts: x-axis, y-axis, z-axis of the drone. Each axis can then be subdivided further into a video processing part, drone movement part, and a collision avoidance part. Figure 2.2 visualizes those 3 axes.

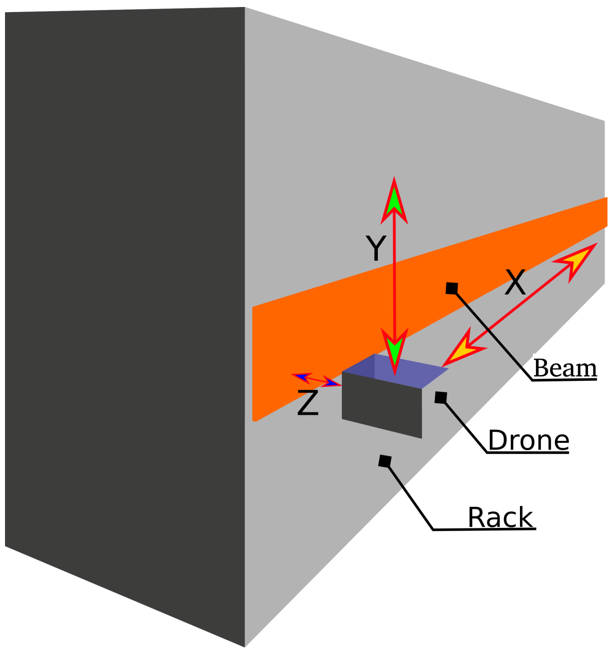


Figure 2.2: Simple drawing visualizing the intended product.

- X-axis:
  - Move the drone sideways
  - Stop the drone at the end of a bar
  - Detect objects in the drone's trajectory
  - Avoid collisions with objects in the drone's trajectory
- Y-axis:
  - Keep the bar in the center of the drone's camera
  - Change the drone's altitude to switch layers
  - Keep track of the current layer and the highest layer
- Z-axis:
  - Estimate the pixel size of the bar
  - Keep the drone to maintain distance from the bar

When referring to "objects", one of the following things is meant:

- Pallets
- Racks
- Forklifts
- People
- Drones

2.3 User Classes and Characteristics

As the product of this iteration will not yet include inventory control functionality, the typical users do currently not include a warehouse employee class. Thus the sole user class of this product will be developers who aim to improve and/or extend the current product. Developers are required to have an extensive knowledge of Python and Opencv.

Chapter 3

Specific Requirements

3.1 Functional Requirements

The following section describes the functions the product shall, should, and might include. For the definitions of these 3 terms refer to the glossary. Each of these 3 terms form a section, which are then further segmented into status, maintenance, and action tasks, for which the definitions can also be found in the glossary. Each functional requirement will have the following format:

[a.b.c][a, a.b.c] Description

Figure 3.1: Format for a functional requirement.

Whereas:

- **a** is the priority group (shall, should, might)
- **b** is the type of task
- **c** is the unique number of that requirement within its group.

Each requirement is prefixed by at least one set of brackets containing its unique ID, but may have a second set containing one or more IDs of prerequisite tasks or groups, each separated by comma. Note that in figure 3.1 the prerequisites bracket contains a single **"a"**, which indicates that all requirements in that priority group are a prerequisite to that task.

[1] The product shall make the drone:

- [**1.1**] Status tasks:
  - [**1.1.1**] Track the vertical center of the beam
  - [**1.1.2**] Estimate the distance from the beam
  - [**1.1.3**] Distinguish objects in the foreground from the background
  - [**1.1.4**][1.1.3, 1.2.1, 1.2.2] Check for objects in its trajectory

- [**1.2**] Maintenance tasks:
  - [**1.2.1**][1.1.1] Maintain its altitude to keep the beam vertically centered
  - [**1.2.2**][1.1.2] Maintain a distance between 30 - 80 centimeters from the beam
- [**1.3**] Action tasks:
  - [**1.3.1**][1.2] Move along the beam from left to right
  - [**1.3.2**][1.1.4] Avoid collisions by landing

[2] The product should make the drone:

- [**2.1**] Status tasks:
  - [**2.1.1**] Keep track of the remaining layers needing to be scanned
  - [**2.1.2**] Determine the end of the beam
- [**2.3**] Action tasks:
  - [**2.3.1**][1.1.2, 2.1] Move to other layers of a rack
  - [**2.3.2**][1.2] Move along the beam from right to left
  - [**2.3.3**][2.1.2] Stop at the end of the beam

[3] The product might make the drone:

- [**3.1**] Status tasks:
  - [**3.1.1**][1.1.4] Determine if an object in its trajectory can be avoided
  - [**3.1.2**] Determine pixel size:distance ratio of the beam
  - [**3.1.3**][1.2.1] Detect bar-codes on the beam
- [**3.2**] Maintenance tasks:
  - [**3.2.1**][3.1.1] Readjust trajectory to avoid collision
- [**3.3**] Action tasks:
  - [**3.3.1**][3.2.1] Move to the opposite rack of the same hallway

3.2 Assumptions

In order for the product to function accordingly, the following assumptions have been made:

- The user is required to fly the drone to the correct starting position. This position is for the drone to face the left end of the lowest beam, at a distance between 30-80 centimeters.
- There are no things sticking out more than 10 centimeters from the rack.
- The drone will have enough battery capacity to complete one side of a rack.
- The warehouse interior lights are turned on and provide enough lighting.
- The drone is equipped with an RGB-camera.
- There is at least 3 meters of free space after the end of a beam.
- The drone remains connected to the device controlling it.

References

Boers, M. (2019), 'Pyav'.
URL: *github.com/mikeboers/PyAV*

Hanyazou (2018), 'Tellopy'.
URL: *github.com/hanyazou/TelloPy*

Lundberg, M. (2019), 'Simple-pid'.
URL: *github.com/m-lundberg/simple-pid*

moses palmer (2019), 'Pynput'.
URL: *github.com/moses-palmer/pynput*

OpenCV Team (2019), 'About opencv'.
URL: *https://opencv.org/about/*

Ryze Robotics (2018), 'Tello user manual'.
URL: *dl-cdn.ryzerobotics.com/downloads/Tello/20180212/Tello+User+Manual+v1.0_EN_2.12.pdf*