Control of UAV for indoor landing in presence of uncertain obstacles

Arne Maenhaut

Supervisors: Prof. dr. ir. Clara-Mihaela Ionescu, Dr. Cosmin Copot
Counsellor: Prof. dr. ir. Clara-Mihaela Ionescu

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Electromechanical Engineering

Department of Electrical Energy, Metals, Mechanical Constructions & Systems
Chair: Prof. dr. ir. Luc Dupré
Faculty of Engineering and Architecture
Academic year 2016-2017

GHENT
UNIVERSITY

Control of UAV for indoor landing in presence of uncertain obstacles

Arne Maenhaut

Supervisors: Prof. dr. ir. Clara-Mihaela Ionescu, Dr. Cosmin Copot
Counsellor: Prof. dr. ir. Clara-Mihaela Ionescu

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Electromechanical Engineering

Department of Electrical Energy, Metals, Mechanical Constructions & Systems
Chair: Prof. dr. ir. Luc Dupré
Faculty of Engineering and Architecture
Academic year 2016-2017

GHENT UNIVERSITY

Permission for Usage

Arne Maenhaut, June 2017

Acknowledgements

When the idea behind my thesis was explained for the first time, I was very excited. It was not only working on a technology that is very tangible, but it was also working on a technology that has a lot of applications. In addition, many things were addressed. This thesis includes, inter alia, sensor fusion, wireless communication, programming in C++, image processing and working with tight constraints as it had to be applied to a real drone. In many cases, it was my first (extensive) acquaintance which made it extra interesting.

As it was "working towards" the development of an autonomous drone, there was a lot of room for creativity. Not just for the way the concept is proven, but also for the proposed methods for the real-world application. It was very pleasant to think conceptually about technology that most likely will be put into use in the next decades.

There was not only left room for creativity, a lot of effort was put in giving (almost instantaneous) feedback on my work and my questions. Therefore, a word of gratitude to prof. dr. ir. Clara-Mihaela Ionescu is definitely in place. Further, I would like to thank dr. Cosmin Copot for pointing out possible difficulties in the real-world application and for sharing his knowledge about the commercially available technology.

Besides that, I also want to thank my friends to provide the necessary relaxation. Last but not least, I would like to thank my parents and girlfriend Elise for their full support and to be there when I need them the most.

Control of UAV for indoor landing in presence of uncertain obstacles

by

Arne Maenhaut

Master's dissertation submitted in order to obtain the academic degree of

Master of Science in Electromechanical Engineering

Academic year 2016-2017

Supervisors: Prof. dr. ir. Clara-Mihaela Ionescu, Dr. Cosmin Copot

Faculty of Engineering and Architecture, Ghent University

Department of Electric Energy, Metals, Mechanical Constructions & Systems

Abstract

In this work, different implementations for a fully autonomous drone are considered. An autonomous drone involves localization, control, obstacle detection, path planning and autonomous landing. First, the options for indoor applications with a low cost drone are discussed and tested, this mainly serves as a proof of concept of the outdoor applications. After this, several options for outdoor applications are discussed.

A robust localization method was implemented using a GPS-like device called Pozyx. This device is especially useful for indoor applications as it a has a high accuracy and can only be used in confined spaces. In order to use Pozyx, some modification had to be made to the drone which caused an imbalance at hovering. This resulted in unavoidable overshoot for the position control.

The obstacle detection was split in two cases: detection of static obstacles and detection of moving obstacles. The (positions of the) static obstacles were detected using an onboard ultrasonic sensor of the drone. For the moving obstacles, an estimation of the position and the velocity were required. As no suitable sensors for the detection of dynamic obstacles were available on the drone, the moving obstacles were marked with colored paper. The obstacles were then detected by color with the bottom camera of the drone.

The path planning methods were not included in the tests done by the author. The information about the methods are the result of research and previous work on this subject at the university. Although the path planning was not the main focus of this thesis, it is still very important in the process of evolving towards a fully autonomous drone. That is why it is added to this work.

The automatic landing algorithms ensures a safe landing as close as possible to a desired location without collision with other obstacles. The experiments pointed out that both methods of obstacle detection offer opportunities in (real-world) outdoor applications.

Keywords

UAV, autonomous, localization, obstacle detection, path planning

Control of UAV for indoor landing in presence of uncertain obstacles

Arne Maenhaut[1]

Supervisor(s): Cosmin Copot[1] and Clara-Mihaela Ionescu[1]

Abstract— **At the moment, the utility of Unmanned Aerial Vehicles is rather limited as trained drone pilots are still needed to control the vehicle. In order to stretch the possibilities, a fully autonomous drone is required. Such a drone requires localization, obstacle detection and avoidance, path planning and autonomous landing. This article starts with exploring solution for indoor applications with a low cost UAV. This is then being evaluated in a lab environment. The indoor evaluation primarily serves as a proof of concept of outdoor applications. Finally, possible solutions for the outdoor applications are proposed based on the acquired insights.**

Keywords—**UAV, autonomous, localization, obstacle detection, path planning**

I. INTRODUCTION

AN autonomous drone is in essence an aerial vehicle that can take off from point A and fly to (and land on) point B without any collisions or human interaction. In order to do so, localization([12], [13], [3], [17]), obstacle detection ([11], [2], [7], [15]), path planning algorithms ([14], [16], [10], [5], [8]) and autonomous landing algorithms ([1], [6], [9]) need to be implemented. For each subproblem a solution is proposed that is executable with the used UAV and that is as much as possible applicable or extendable to the outdoor application.

II. METHODS

A. Localization

In order to have a robust position estimation, a localization method similar to GPS is applied. For the indoor experiment a device is used that is called Pozyx. Pozyx functions according to the same principles as GPS, but it achieves a higher accuracy by using ultra-wideband technology and it is applied on a much smaller scale. To reduce the noise and to have a better position estimation, the information from the Pozyx is fused with the information from the inertial measuring unit by using a Kalman filter.

B. Control

Initially, controllers of previous work were considered. Unfortunately, the controllers did not give the desired result. That is why a heuristic method was used to design a PD controller. In the experiments some overshoot is noticeable, this is mainly due to the imbalance of the drone when hovering. Due to the imbalance, the drone is very unstable which cannot be completely compensated by the controller. However, the controlling was sufficient to show the desired proof of concept.

[1] DySC, Ghent University Technology Park 914 B-9052 Zwijnaarde

C. Obstacle Detection

Obstacle detection of static obstacles is implemented by using the ultrasonic sensor of the drone. Obstacles on the ground are detected by comparing the absolute height (h_a) to the relative height (h_r) as shown in figure 1. When the difference exceeds a threshold value, one can assume that an obstacle is present on that location.

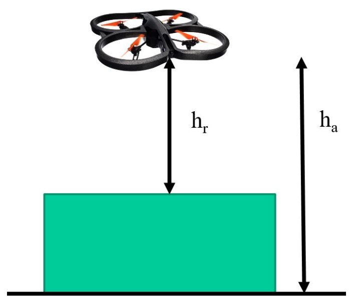


Fig. 1. Obstacle detection of static obstacles

The obstacle detection of moving obstacles uses the bottom camera of the drone. Obstacles can be detected based on color because the obstacles are marked with colored paper. First, the colored image is converted to a binary image. In this image, each pixel indicates whether an obstacle is present on that location. The conversion is done by specifying a range of colors that corresponds to the color of the marker (paper). Once the position of the obstacle is known in pixel coordinates, the coordinates are converted to world coordinates. The velocity of the object can then be estimated by taking the finite difference of the position.

D. Path Planning

The path planning methods discussed in this article are mainly deduced from the results of Diericx in “Evaluation of path planning algorithms for a quadcopter during indoor flight”[4]. In literature, graph-based search or potential fields are used in most cases for path planning.

D.1 Graph-based search

In this method the environment is discretized. In this way it is possible to apply a shortest path algorithm. All methods used in this article are based on Dijkstra’s algorithm. Costs are defined between different nodes. The composition of these costs depends on the preferred behavior (e.g. fast path or safe path).

D.2 Potential fields

Potential fields applies repelling forces from obstacles and attractive forces towards the target location. This method is known for its natural movement of the agent. The difficulty with this method lies in preventing the agent from getting stuck in a local minimum.

E. *Autonomous Landing*

To find the optimal spot to land, the algorithm needs to take into account all obstacles on the ground and the dimensions of the agent itself. From this data, the algorithm calculates the spot that is as close as possible to the target location without danger of collision with obstacles.

III. Results

The experiments show that an autonomous landing in presence of static obstacles can be performed using distance meters. In figure 2, a graphical representation can be found of the environment. Figure 3 shows the map of the environment that is internally stored in the drone. As can be seen, the map matches the real situation. Deviations are mainly due to the used algorithm.

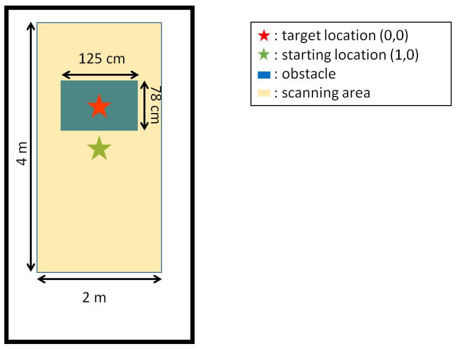


Fig. 2. Graphical representation environment

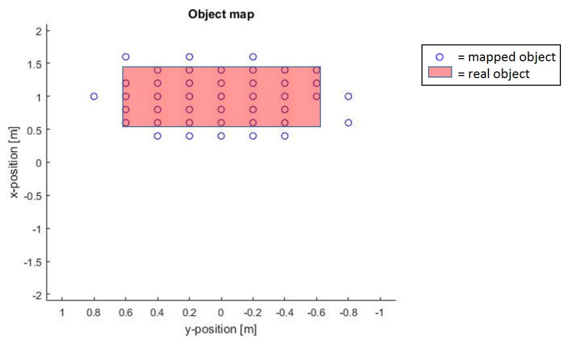


Fig. 3. Environment mapped by the UAV

The experiments with dynamic obstacles show that velocity estimation based on vision is feasible from a drone. In the experiments, a Lego Mindstorm robot drove under the drone with a speed of 0.2 m/s. The velocity estimations with a moving average filter of 1.5 s resulted in values between 0.185 and 0.240 m/s.

IV. Conclusion

In this work, some conceptual ideas for an autonomous drone were proven. Furthermore, possible solutions for the real-world application were provided. This work can be used as a guideline to evolve towards the development of a prototype of an autonomous drone. Further research can focus more on the required features of the technological components of the UAV and on the development of algorithms that can deal with complex environments that include both human-controlled dynamic obstacles and non-human-controlled dynamic obstacles.

References

[1] Roman Barták, Andrej Hrasko, and David Obdrzálek. “On Autonomous Landing of AR. Drone: Hands-On Experience.” In: *FLAIRS Conference*. 2014.

[2] Jeffrey Byrne, Martin Cosgrove, and Raman Mehra. “Stereo based obstacle detection for an unmanned air vehicle”. In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE. 2006, pp. 2830–2835.

[3] Andrew J Davison et al. “MonoSLAM: Real-time single camera SLAM”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007).

[4] Bram Dierickx. “Evaluation of path planning algorithms for a quadcopter during indoor flight”. Master thesis. Ghent University, 2014-2015.

[5] Dave Ferguson and Anthony Stentz. “Field D*: An Interpolation-based Path Planner and Replanner”. In: *Springer Tracts in Advanced Robotics* 28 (2007).

[6] Pedro J Garcia-Pardo, Gaurav S Sukhatme, and James F Montgomery. “Towards vision-based safe landing for an autonomous helicopter”. In: *Robotics and Autonomous Systems* 38.1 (2002), pp. 19–29.

[7] Stefan Hrabar et al. “Combined optic-flow and stereo-based navigation of urban canyons for a UAV”. In: *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE. 2005, pp. 3309–3316.

[8] Yong K Hwang and Narendra Ahuja. “A potential field approach to path planning”. In: *IEEE Transactions on Robotics and Automation* 8.1 (1992), pp. 23–32.

[9] Andrew Johnson, James Montgomery, and Larry Matthies. “Vision guided landing of an autonomous helicopter in hazardous terrain”. In: *Robotics and automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE international conference on*. IEEE. 2005, pp. 3966–3971.

[10] Geoff Gordon Anthony Stentz Maxim Likhachev Dave Ferguson and Sebastian Thrun. “Anytime Dynamic A*: An Anytime, Replanning Algorithm”. In: *Autonomous Robots* (2005).

[11] Tim G McGee, Raja Sengupta, and Karl Hedrick. “Obstacle detection for small autonomous aircraft using sky segmentation”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 4679–4684.

[12] Daniel Mellinger and Vijay Kumar. “Minimum snap trajectory generation and control for quadrotors”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2520–2525.

[13] Harold F. Murcia. “A quadrotor as remote sensor for on-line prole measurement during harvesting process”. Master thesis. Ghent University, 2013-2014.

[14] N. J. Nilsson. “Principles of Artificial Intelligence”. In: *Autonomous Robots* (1980).

[15] Roberto Sabatini, Alessandro Gardi, and M Richardson. “LIDAR obstacle warning and avoidance system for unmanned aircraft”. In: *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering* 8.4 (2014), pp. 718–729.

[16] Maxim Likhachev Sven Koening. “D* Lite”. In: *AAAI-02 Proceedings* (2002).

[17] Chang-Sun Yoo Chang-Sun Yoo and Iee-Ki Ahn Iee-Ki Ahn. “Low cost GPS/INS sensor fusion system for UAV navigation”. In: *Digital Avionics Systems Conference, 2003. DASC’03. The 22nd*. Vol. 2. IEEE. 2003, 8–A.

Contents

Used abbreviations

AED	Automated External Defibrillator
API	Application Programming Interface
CPU	Central Processing Unit
DDR	Double Data Rate
GPS	Global Positioning System
HSV	Hue Saturation Value
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
LIDAR	LIght Detection And Ranging
PD	Proportional-Derivative
PTAM	Parallel Tracking and Mapping
PWM	Pulse-Width Modulation
QVGA	Quarter Video Graphics Array
RAM	Random-Access Memory
RGB	Red Green Blue
ROS	Robot Operating System
SDHC	Secure Digital High Capacity
SDK	Software Development Kit
SLAM	Simultaneous Localization And Mapping
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
UWB	Ultra-WideBand
WiFi	Wireless Fidelity

List of Figures

List of Tables

Chapter 1

Introduction

Sudden cardiac arrests are responsible for nearly 700 000 deaths each year in Europe[35]. In case of a “shockable” cardiac arrest, a survival rate above the 70 percent can be achieved when defibrillation is done within the first 5 minutes after collapse [28]. Each minute of delay reduces the chance of survival by 10-12%. To be faster on the spot and to provide a better and faster assistance, the idea came to use an UAV that is equipped with an AED. First studies confirm that drone networks have the potential to reduce the travel time of an AED towards the victim greatly with limited costs[32].

1.1 Background

Like many other devices, the UAV was originally used in military applications. Nowadays UAVs are not only used in the military anymore, but they also serve for the filming industry, agriculture sector, surveillance and many other purposes. For some applications, full autonomy is required as it reduces the necessary manpower.

Ambulance drone

In 2014, Living Tomorrow Belgium has financed a master thesis in partnership with TU Delft and Ghent University Hospital in order to design an “ambulance drone” for prevention of cardiac arrests[25]. This was a design-driven process that led to a prototype that conceptualized the idea. This first prototype did not meet the functional requirements. To further develop this prototype, a requirement-driven approach is recommended. By exploring the possibilities to meet the requirements with available technological components, it will be possible to build a true prototype that is suitable for real-world testing.

Figure 1.1: Ambulance drone[1]

Amazon Air Prime

In 2016, Amazon, world's largest online retailer, introduced a drone-based delivery system that is called Amazon Prime Air[2]. They aspire a delivery system that get packages in a safe way to customers in 30 minutes. In this way the efficiency of the transporting system and the services to customers will be greatly enhanced. Currently the drones are still in the testing phase.

Figure 1.2: Package delivery drone from Amazon[3]

[1]Figure from "Drones for good"[25]
[2]Amazon Prime Air: https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011
[3]Figure from *Amazon Prime Air*[2]

Rather than using an UAS as a logistics solution, this thesis focuses on an UAS that deals with more complex situations. Amazon intends to let the drone land on a landing pad that must meet a number of conditions regarding available space around the landing platform, while this thesis intends to develop an UAS that includes automated safe landing in complex contexts which include small streets, cars, trees and much more other obstacles.

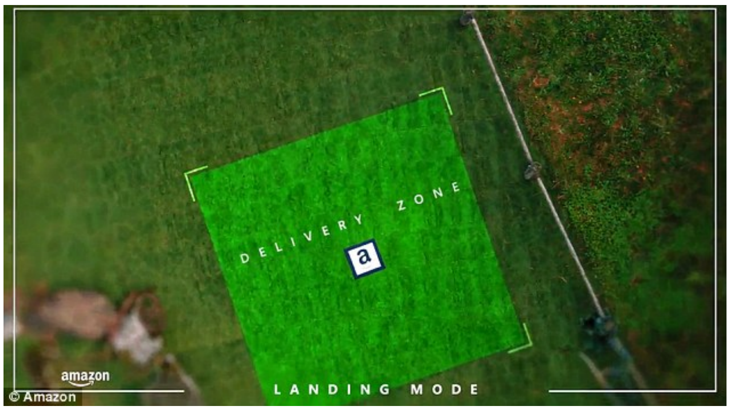


Figure 1.3: Landing pad Amazon Air Prime[4]

1.2 Problem context

While drones have been commercially available for a while, no fully autonomous drone has yet been developed. Such an autonomous drone has many requirements. An autonomous drone needs to be able to take off from point A and fly to and land on point B in a fast and safe way without any collisions. In addition to the costs and the currently limited flight time, one of the biggest challenges is the obstacle detection and the controlling of the drone. The application does not only require to sense and avoid static obstacles like buildings, but also dynamic obstacle like birds.

Current technical restrictions

Although much progress has been made regarding obstacle detection, to this day there is no standard method to detect obstacles with small UAVs that is affordable and robust. The application does not only require an accurate detection of static obstacles, but also the detection of dynamic obstacles. If it is also taken into account that the hardware of the drone needs to be implemented on an UAV, which results in a high restriction of the weight and the power consumption of the hardware, one can realize that this problem is challenging.

[4]Figure from *Jeremy Clarkson reveals new prototype drones for Amazon Prime Air deliveries*[18]

A robust localization method is provided by GPS. It is the question if the GPS signal is accurate enough for an autonomous UAV. When information of static obstacles (e.g. buildings) is taken from a database, the accuracy of the GPS only influences the accuracy of the location of the UAV. When information of static obstacles (e.g. parked cars) is taken from a mapping done by the drone itself, the accuracy of the GPS not only affects the accuracy of the location of the drone but also the accuracy of the mapping. Since the accuracy of the GPS signal is a few meters, a high safety margin will be required. In order to fly in small streets and to obtain a higher freedom of movement, a more precise localization method will be required.

1.3 Objective of this thesis

This thesis primarily focuses on the sense and avoid requirements of the UAV and on a proper algorithm that finds a safe spot to land. It does not focus on the practical problems such as shock/water resistance of the UAV, UAS regulations or communication options. It mainly serves as a proof of concept of a drone that can go from point A to point B without any collisions with obstacles. It gives a basic idea of the requirements of the drone, provides some algorithms to tackle certain problems and gives some insight in the possible problems that could be encountered in the real-world application. As the connection of additional sensors is not possible (due to weight restrictions), the drone has limitations and the proof of concept is executed indoors, some sacrifices must be made concerning the direct applicability in reality. However, only solutions are chosen that are similar to solutions for the real-world applications such that they are easily extendable to reality. Of course not every problem can get thoroughly tackled because of certain limitations. In this case, a simplified solution is proposed. To thoroughly tackle these problem, further research is necessary.
The remark should be made that although the theme of this thesis is to help the problem of cardiac arrests, many of the acquired insights can be extended to other domains.

1.4 Structure of the thesis

This thesis shows the basic requirements of an autonomous drone. It is not limited to a description of hardware and software implementations, but it also provides a proof of concept that is performed indoors.

The thesis is structured as follows:

Chapter 1 describes the motivation, the problem and the approach of the thesis.
Chapter 2 contains an overview of contributions of other authors to this subject.
Chapter 3 gives a description of the used hardware and software.
Chapter 4 explains how the drone is able to provide an accurate and robust position estimation.

Chapter 5 describes the used controller for the experiment.
Chapter 6 shows the obstacle detection techniques used for the indoors experiment and provides possible obstacle detection methods that can be used in the real-world (outdoor) application.
Chapter 7 gives an overview of possible path planning methods. It also discusses the results of previous experiments performed on the department.
Chapter 8 describes the used landing algorithms in the experiments.
Chapter 9 shows the result of the experimental validation.
Chapter 10 contains the overall conclusions and recommendations for future work.

Chapter 2

State of the Art

In this chapter, solutions that are provided by other authors are examined. Since many things are addressed, the problems are subdivided into subproblems.

2.1 Localization

In literature, different methods that are suitable for drone localization can be found.

Localization methods for lab environments

Localization is possible using external tracking systems as is applied in “On-board and ground visual pose estimation techniques for UAV control”[21]. Although the use of this method is restricted to lab environments, it can be useful in the development process of prototypes.
Diericx in “Evaluation of path planning algorithms for a quadcopter during indoor flight”[6] and Murcia in “A quadrotor as remote sensor for on-line prole measurement during harvesting process”[26] used odometry fused with a localization method using visual ground patterns. The patterns on the ground are used to calibrate the position from odometry as odometry is prone to drifting.

Real-world localization methods

Another frequently used localization method is Simultaneous Localization And Mapping. This algorithm constructs, as the name suggests, a map of the environment and provides an estimation of the location of the agent at the same time. Davison et al. present a real-time SLAM algorithm using a single camera to do position estimations in “MonoSLAM: Real-time single camera SLAM”[5].
A localization method for indoor applications that is based on the same principles as GPS, is described in “WiFi ad-hoc mesh network and MAC protocol solution for UWB indoor localization systems”[34]. It has a higher accuracy than GPS but also a lower range. The best known outdoor localization method is GPS, whether or not fused with INS as described by Yoo and

Ahn in “Low cost GPS/INS sensor fusion system for UAV navigation”[42].
A promising technology addressed to this subject, is Project Tango.[1]. The idea is to let devices estimate their position with respect to the environment in a similar way as humans know their position with their eyes.[11] In order to do this, it fuses information from motion tracking, area learning and depth perception.

2.2 Obstacle Detection

Obstacle detection is a key functionality of an autonomous drone. As a proper detection of obstacles has many requirements, it is also one of the biggest challenges in the development process.

Detection based on the visible spectrum

McGee, Sengupta, and Hedrick explored a computationally inexpensive method for obstacle detection and avoidance by using computer vision in “Obstacle detection for small autonomous aircraft using sky segmentation”[23].
A more computationally expensive method is stereo based detection. Stereoscopic vision combines the information of cameras at different points of view to render a 3D image. Research has been done in this regard in “Stereo based obstacle detection for an unmanned air vehicle”[4]. Stereo vision can be combined with optical flow information. The optical flow can, among other things, help the UAV to stay centered in urban canyons. Research on this topic is done in “Combined optic-flow and stereo-based navigation of urban canyons for a UAV”[16].

LIDAR-based obstacle detection

As it is a very intuitive way for humans to detect obstacles by vision, the above described obstacle detection methods are extensively explored. These methods have the big disadvantage that they are not suitable in the dark. To be able to detect obstacles in the dark, devices that work with other portions of the electromagnetic spectrum can be used. A possibility is to scan the environment point by point with a LIDAR-based device as it is described in “LIDAR obstacle warning and avoidance system for unmanned aircraft”[36]. This method estimates the distance to obstacles using the difference in return times of laser pulses.
Next to systems that scan the environment point by point, methods exist that can capture the entire scene with each laser pulse. One of those devices is called time-of-flight cameras. These devices are very suitable for dynamic environments. When using the Doppler effect, these devices can quickly provide an estimation of the speed of moving obstacles in addition to the position estimation of obstacles. This is being addressed in “Doppler time-of-flight imaging”[13].
Another device that can detect obstacles, is the structured-light 3D scanner. This device

[1]https://get.google.com/tango/

projects a known pattern on obstacles. The deformation of the pattern then allows vision systems to calculate the distance to objects and to give additional information of the surface. Research on this topic has been done in "Structured-light 3D surface imaging: a tutorial"[10]. However, the range of this device is too small for obstacle detection from an UAV.

2.3 Path Planning

A fully autonomous drone requires an algorithm that generates a path to go from point A to point B. The most important requirement of the path is that it has to be collision-free. A lot of path-planning algorithms already exist, the optimal algorithm depends on the specification of the optimal path.

Graph-based path planning

A first approach for path planning is based on Dijkstra's algorithm. The idea is to discretize the environment. In this way it is possible to apply a shortest path algorithm in order to find an optimal path.
In literature, improvements to Dijkstra's algorithm can be found. One of the improvements is called A* which is described in "Principles of Artificial Intelligence"[27]. This algorithm converges faster to a solution by making use of a heuristic. Depending on the situation, it is possible to implement the algorithm with a lower computation time or with a higher certainty that the calculated path is optimal.
This algorithm recalculates the path each time a new obstacle appears. In "D* Lite"[38] an algorithm is introduced that addresses this problem by using the previously calculated graph to calculate the path.
Anytime Dynamic A* is an algorithm that is introduced in "Anytime Dynamic A*: An Anytime, Replanning Algorithm"[22] by Maxim Likhachev and Thrun. It is a combination of an anytime algorithm which focuses on finding a solution very quickly and a replanning algorithm which copes with incomplete information and changing environments.
Ferguson and Stentz propose an algorithm (Field D*) in "Field D*: An Interpolation-based Path Planner and Replanner"[8] which is not constrained to discrete states. It applies linear interpolation to find a more optimal path.

Potential fields

Potential fields is another approach for path planning. It applies repelling forces from obstacles and attractive forces towards the target location. It results in a more natural and smoother movement of the agent. Potential fields is, inter alia, described in "A potential field approach to path planning"[17] and "New potential functions for mobile robot path planning"[9].

Dynamic obstacles

Collision avoidance of dynamic obstacles is very important during flight and should be integrated into the path planning. Research[24][41][33] has been conducted on path planning algorithms that deal with dynamic obstacles. However, no globally accepted solution is available.

2.4 Other

Controller

Researchers applied different control strategies to the Parrot AR.Drone 2.0. Hernandez et al. from UGhent implemented a PID and internal model controller in "Identification and path following control of an AR. Drone quadrotor"[14] and applied model predictive control in "Towards the development of a smart flying sensor: Illustration in the field of precision agriculture"[15]. In "Output feedback control of a quadrotor UAV using neural networks"[7] a nonlinear controller is proposed using neural networks. The neural networks are used to learn the complete dynamics of the UAV.

Autonomous landing

Autonomous landing is described in "On Autonomous Landing of AR. Drone: Hands-On Experience."[3] by Barták, Hrasko, and Obdrzálek. It uses landing patterns to find the spot to land. In "Vision guided landing of an autonomous helicopter in hazardous terrain"[19] a vision-based landing algorithm is described. The algorithm uses a single moving camera to generate a cloud of 3D points and to construct a terrain map in this way.

Prototyping of autonomous UAVs

The development of an autonomous UAV is a complex matter. Next to the strict regulations, outdoor testing of prototypes in real-world situations can be very dangerous. Although various tests can be performed indoors with self-established obstacles, the testing environment will be lacking complexity at some stage. An interesting idea regarding this topic is described in "Measurable Augmented Reality for Prototyping Cyberphysical Systems: A Robotics Platform to Aid the Hardware Prototyping and Performance Testing of Algorithms"[29]. In this article, a projected environment is proposed to simulate real-world environments. Through augmented reality, very complex situations, which includes dynamic environments and various virtual vehicles, can be created at low cost and danger.

Chapter 3

Hardware and Software Setup

3.1 Lego Mindstorms EV3

Lego Mindstorms are an extension of the well known Lego bricks. The Mindstorms consist of an intelligent computer which can be connected to electric motors and sensors.

3.1.1 Hardware

In this subsection, an overview of the used hardware is given with the most important specifications.

- EV3 brick
  - WiFi
  - Bluetooth

Figure 3.1: EV3 brick[1]

- Large EV3 motors
  - Tacho feedback to one degree of accuracy
  - 160-170 rpm

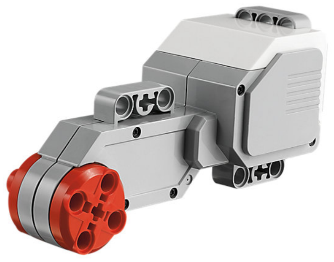


Figure 3.2: Large EV3 motor[1]

- Ultrasonic sensor
  - Measures distance in a range of 1-250 cm
  - Accuracy of 1 cm

Figure 3.3: Ultrasonic sensor[1]

- Gyro sensor
  - Accuracy of 3 degrees
  - Maximum output of 440 degrees/second
  - Sample rate of 1 kHz

Figure 3.4: Gyro sensor[1]

3.1.2 Software

Instead of the standard EV3 firmware, leJOS[2] is used. LeJOS is a firmware replacement that makes it possible to write programs in Java. The software already provides a high level API, in this way it is possible to quickly write basic programs.

3.2 Parrot AR.Drone 2.0

The AR.Drone 2.0 is a remote controlled quadcopter build by the French company Parrot. The drone can be controlled by Android and iOS. Parrot also launched the AR.Drone API for developers. Due to the affordability and the open platform, the AR.Drone gained a lot of interest from researchers.

3.2.1 Hardware

The drone has an internal controller that regulates the current of each PWM motor for given speed setpoints in each direction (x, y and z). It is advantageous that the drone already hovers stable from itself, but the downside is that it is not possible to gain insight in the internal controller or to control each blade separately.

[1]Figure from *Lego shop*[20]
[2]leJOS: http://www.lejos.org

- WiFi
- 3 axis accelerometer with 50 mg precision
- 3 axis gyroscope with 2000°/second precision
- Pressure sensor with 10 Pa precision
- 40kHz Ultrasonic sensor
- Downward facing camera: QVGA (320x240) @ 60 fps
- Front camera: 720p (1280x720) @ 30 fps (92° wide angle lens)

Figure 3.5: Parrot AR.Drone 2.0[3]

3.2.2 Software

During the flight, the drone communicates with an external computer via WiFi. This computer processes the sensor information acquired from the drone and sends appropriate commands back to the drone. The software[4] for the processing of the sensor signals and the communication with the drone is written in C++, it includes the AR.Drone 2.0 SDK[5] and OpenCV[6] library. The SDK includes some basic functionalities like giving a speed setpoint and receiving information from sensors. OpenCV is an open source computer vision and machine learning library. It is used in this dissertation to detect obstacles. There was explicitly chosen to not use software like ROS [7] which already provides algorithms for PTAM and state estimation.

[3]Figure from *Parrot AR.Drone 2.0 elite edition*[30]
[4]CV Drone: https://github.com/puku0x/cvdrone
[5]AR.Drone 2.0 SDK: http://developer.parrot.com/docs/SDK2/ARDrone_SDK_2_0_1.zip
[6]OpenCV: http://opencv.org/
[7]ROS package for AR.Drone 2.0: http://wiki.ros.org/tum_ardrone

Chapter 4

Localization Methods

In order to reach a victim, an accurate position estimation is needed. In outdoor applications GPS usually provides an estimation that is accurate enough. For indoor applications GPS is not accurate enough. The options for indoor position estimation are: visual navigation, the onboard IMU data or a positioning system comparable with GPS but with a higher accuracy. In order to provide indoor solutions that are comparable to outdoor solutions, there is opted to only use the last two options for the state estimation.

4.1 IMU data processing

The inertial measuring unit measures the roll, pitch and yaw angle. The estimations are obtained by fusing the information from a 3-axis gyroscope with the information from a 3-axis accelerometer and a magnetometer. The velocities are internally estimated by the optical flow from the bottom camera. Position estimations can then be obtained by integrating the velocities. As the velocities are given in drone coordinates, a coordinate transformation to real world coordinates is required.

$$x_k = x_{k-1} + \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot V_{drone} \cdot \Delta t \tag{4.1}$$

A better estimation of the height can be obtained by fusing the velocity in z-direction with the information from the ultrasonic sensor. The estimation can be further improved by applying a Kalman filter. This method has the big disadvantage that an error in position estimation propagates. Testing pointed out that this estimation is indeed very prone to drifting and that the drifting happens very fast. However, this is no reason to completely get rid of this method as it is a good source of information for higher frequencies.

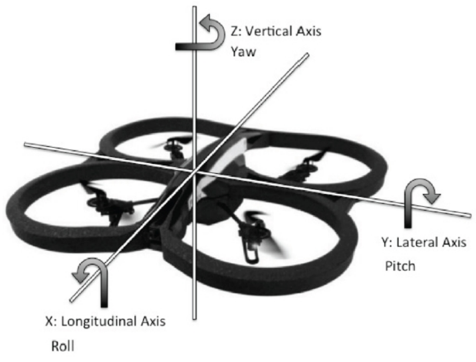


Figure 4.1: Measured angles from the IMU[1]

4.2 Pozyx

Pozyx is an indoor positioning system that is based on the same principles as GPS but on a much smaller scale. A far more accurate position estimation (10 cm error) is obtained by using ultra-wideband technology.

Principle

Some anchors (at least 3) are placed in a room. The anchors should not be placed on one line or on one plane. The absolute position of these anchors needs to be known. By measuring the distance between the Pozyx tag and each anchor, the position of the Pozyx tag can be estimated.

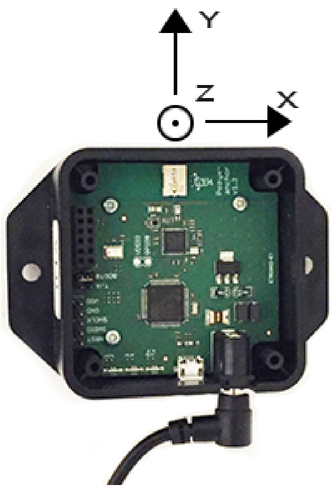


Figure 4.2: Pozyx anchor[2]

[1]Figure from "The use of gaze to control drones"[12]

4.2.1 Tackling

In practice some outliers in the data were noticed. These were filtered out by defining a maximum allowed deviation of the current position compared to the previous one. When the deviation exceeds the threshold value, the current position is set equal to the previous one. Although the threshold value is chosen quite high, a limit is set to consecutive changes of the value of the current position to the value of the previous position. A limitation is set to avoid false positive outliers and to be robust. The filter is implemented in C++ as follows:

```
//positions in mm
if (abs(xpoz - x_poz_prev) > 500 && x_over_lim_counter <12) {

                        xpoz_filter = x_poz_prev;
                        x_over_lim_counter = x_over_lim_counter + 1;

                }
                else {
                        x_over_lim_counter = 0;
                        xpoz_filter = xpoz;
                }
        x_poz_prev=xpoz_filter;
```

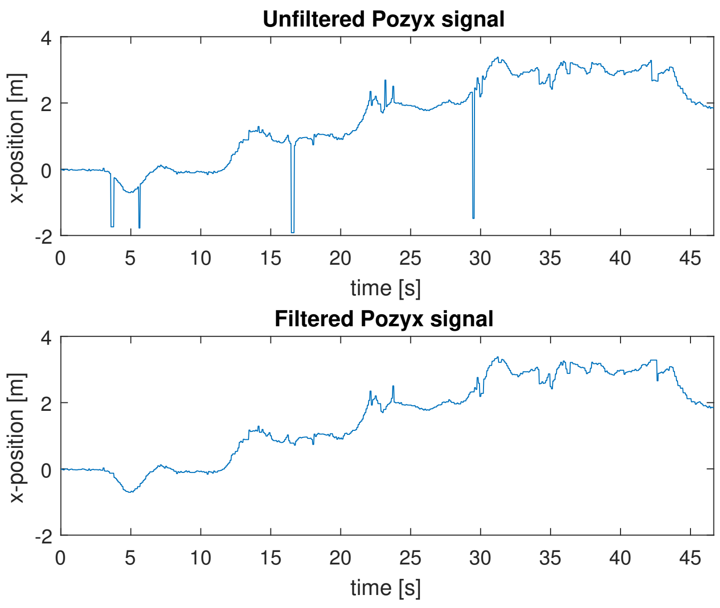


Figure 4.3: Outliers Pozyx

[2]Figure from *Pozyx accurate positioning*[31]

The Pozyx offers a robust position estimation. However, this signal is noisy so that it is not recommended to use the Pozyx as the only source of information.

4.3 Sensor fusion method

To combine the benefits of the IMU and the Pozyx, there is opted to use a Kalman filter to fuse these signals. A kalman filter combines the information of different sensors and the expected state from the physical model to estimate the state.

$$x = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix} \tag{4.2}$$

For the transition matrix A a constant velocity model was chosen. The variable dt is equal to the mean execution time of the loop in equation 4.3. With the used sensors it is possible to measure the states directly, so the measurement matrix C is equal to a unity matrix.

$$A = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

In order to design a Kalman filter, the process noise covariance Q and the measurement noise covariance R are needed. The process noise covariance matrix and the covariances of the velocity measurements were already determined. The covariances of the Pozyx were then tuned manually until the desired result was obtained.

$$Q = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 \end{bmatrix} \tag{4.5}$$

$$R = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05 \end{bmatrix} \tag{4.6}$$

A Kalman filter consists of two phases: the prediction phase and the update phase. In the prediction phase the Kalman filter estimates the position and its uncertainty based on the physical model.

$$\hat{x}_k = A \cdot \hat{x}_{k-1} \tag{4.7}$$

$$P_k = A \cdot P_{k-1} \cdot A^T + Q \tag{4.8}$$

In the update phase the observed measurements are used to find a new best estimate.

$$K' = P_k \cdot C^T (C \cdot P_k \cdot C^T + R)^{-1} \tag{4.9}$$

$$\hat{x}'_k = \hat{x}_k + K'(z_k - C \cdot \hat{x}_k) \tag{4.10}$$

$$P'_k = P_k - K' \cdot C \cdot P_k \tag{4.11}$$

4.3.1 Comparison of the different localization methods

Figure 4.4 shows an estimation of the position with the Kalman filter, Pozyx and the IMU. In reality the drone starts at $x \approx 0\ m$, then goes around 21 seconds to $x \approx 1\ m$ and finally goes around 30 seconds to $x \approx 2\ m$. It is clear that it is not possible to only use the IMU for position estimation as it starts to drift very quickly. The Pozyx has a robust position estimation but it is noisy. The sensor fusion combines the advantages of both signals. It is a robust estimation and it is not noisy.

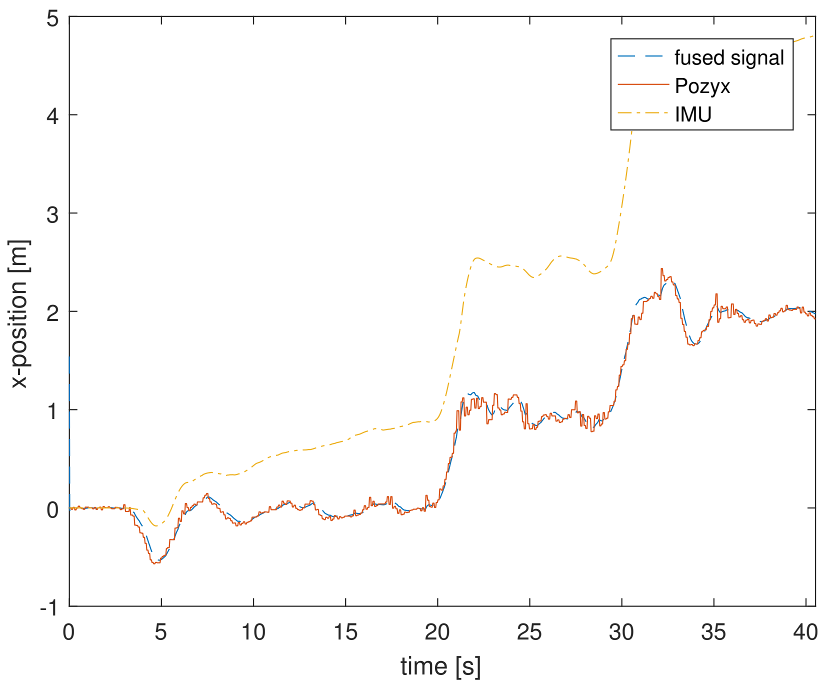


Figure 4.4: Comparison of the different localization methods

Chapter 5

Design of the controller

In this chapter the controller implemented in the drone is discussed.

5.1 Introduction

The application requires position control of the drone. The internal controller of the drone receives speed setpoints in all directions as input. A PD controller is chosen for the position control. When a perfect internal velocity controller is considered, the internal controller combined with the dynamics can be modeled as $\frac{1}{s}$. This shows intuitively that the D-action is required and the I-action is actually unnecessary. Initially, controllers from previous work[26] were considered, but they did not result in a proper control.

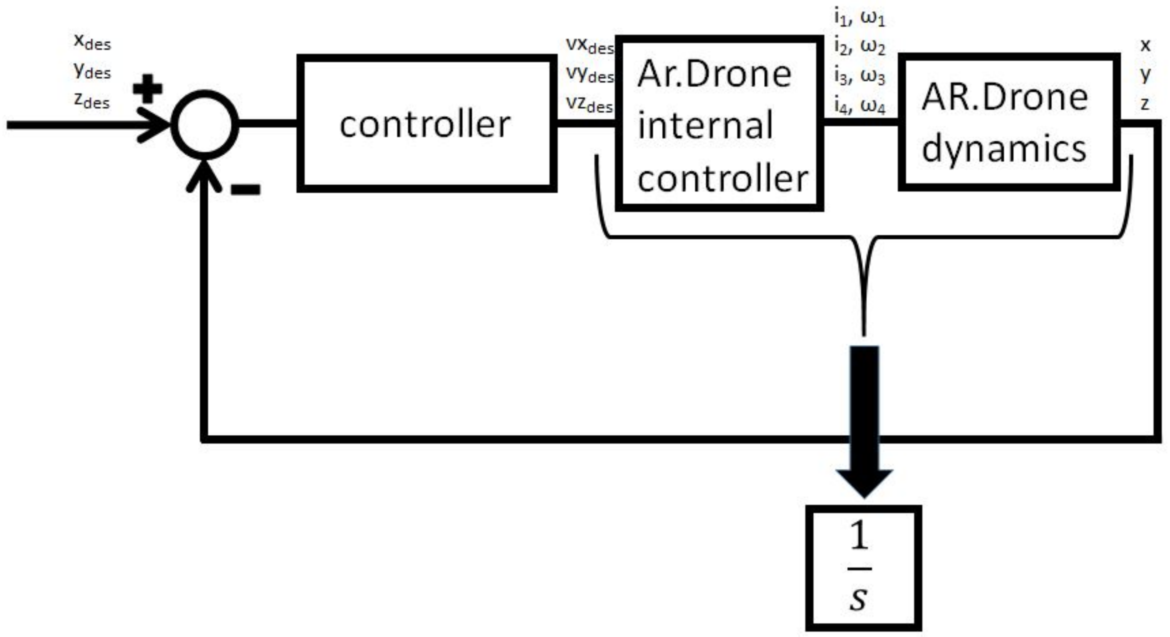


Figure 5.1: Control with perfect internal controller

5.2 Implementation of the PD controller

The controller is implemented as follows:

$$e_x = x_{set} - x \tag{5.1}$$

$$e_y = y_{set} - y \tag{5.2}$$

$$e_z = z_{set} - z \tag{5.3}$$

Velocity inputs in absolute coordinates:

$$v_{x,abs} = K_{p,x} \cdot (e_x + T_{d,x} \cdot \frac{e_x - e_{x,prev}}{dt}) \tag{5.4}$$

$$v_{y,abs} = K_{p,y} \cdot (e_y + T_{d,y} \cdot \frac{e_y - e_{y,prev}}{dt}) \tag{5.5}$$

$$v_{z,abs} = K_{p,z} \cdot (e_z + T_{d,z} \cdot \frac{e_z - e_{z,prev}}{dt}) \tag{5.6}$$

The control efforts in relative (drone) coordinates:

$$v_{x,rel} = \cos(yaw) \cdot v_{x,abs} + \sin(yaw) \cdot v_{y,abs} \tag{5.7}$$

$$v_{y,rel} = -\sin(yaw) \cdot v_{x,abs} + \cos(yaw) \cdot v_{y,abs} \tag{5.8}$$

$$v_{z,rel} = v_{z,abs} \tag{5.9}$$

5.3 Smoothing the control effort

Taking the derivative increases the noise. To lower the noise in the control effort, a smoother D-action is desired. Currently, the D-action of the effort is defined as follows:

$$e_{d,x} = K_{p,x} \cdot T_{d,x} \cdot \frac{e_x - e_{x,prev}}{dt} \tag{5.10}$$

$$e_{d,y} = K_{p,y} \cdot T_{d,y} \cdot \frac{e_y - e_{y,prev}}{dt} \tag{5.11}$$

$$e_{d,z} = K_{p,z} \cdot T_{d,z} \cdot \frac{e_z - e_{z,prev}}{dt} \tag{5.12}$$

The efforts are equal to:

$$e_x = x_{set} - x \tag{5.13}$$

$$e_y = y_{set} - y \tag{5.14}$$

$$e_z = z_{set} - z \tag{5.15}$$

This means that equations 5.10, 5.11 and 5.12 can be rewritten as follows:

$$e_{d,x} = K_{p,x} \cdot T_{d,x} \cdot \frac{(x_{set} - x_{set,prev}) - (x - x_{prev})}{dt} \tag{5.16}$$

$$e_{d,y} = K_{p,y} \cdot T_{d,y} \cdot \frac{(y_{set} - y_{set,prev}) - (y - y_{prev})}{dt} \tag{5.17}$$

$$e_{d,z} = K_{p,z} \cdot T_{d,z} \cdot \frac{(z_{set} - z_{set,prev}) - (z - z_{prev})}{dt} \tag{5.18}$$

Since the position setpoints in the application are not constantly changed, but only in discrete intervals (when a waypoint is reached), the influence of the difference of the current setpoint and the previous setpoint can be neglected.

Then the following substitution can be done:

$$v_{x,abs} = \frac{x - x_{prev}}{dt} \tag{5.19}$$

$$v_{y,abs} = \frac{y - y_{prev}}{dt} \tag{5.20}$$

$$v_{z,abs} = \frac{z - z_{prev}}{dt} \tag{5.21}$$

This results in following D-actions:

$$e_{d,x} = -K_{p,x} \cdot T_{d,x} \cdot v_{x,abs} \tag{5.22}$$

$$e_{d,y} = -K_{p,y} \cdot T_{d,y} \cdot v_{y,abs} \tag{5.23}$$

$$e_{d,z} = -K_{p,z} \cdot T_{d,z} \cdot v_{z,abs} \tag{5.24}$$

The comparison between the old definition of the D-action and the new definition can be found in figure 5.2.

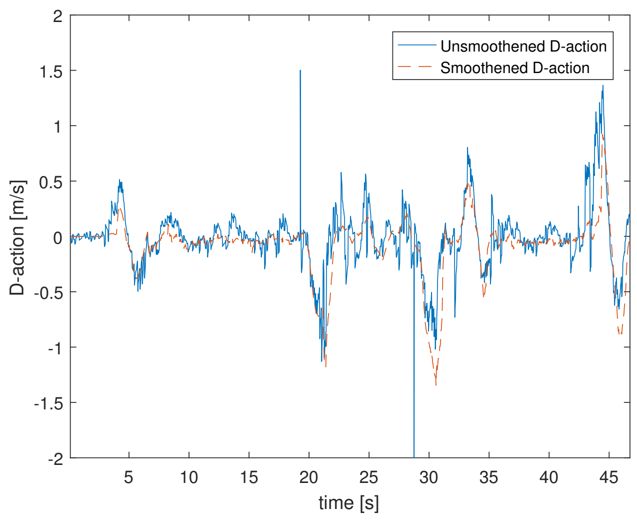


Figure 5.2: Unsmoothened and smoothened D-action of position control

5.4 Ziegler-Nichols tuning method

The Ziegler-Nichols tuning rule is a heuristic method to determine the parameters of a PID controller. This method starts with putting K_p equal to zero. K_p is then increased until the ultimate gain K_u is reached. This is the case when the output shows stable oscillations. The ultimate gain K_u together with the oscillation period T_u then determines the value of K_p and T_d.

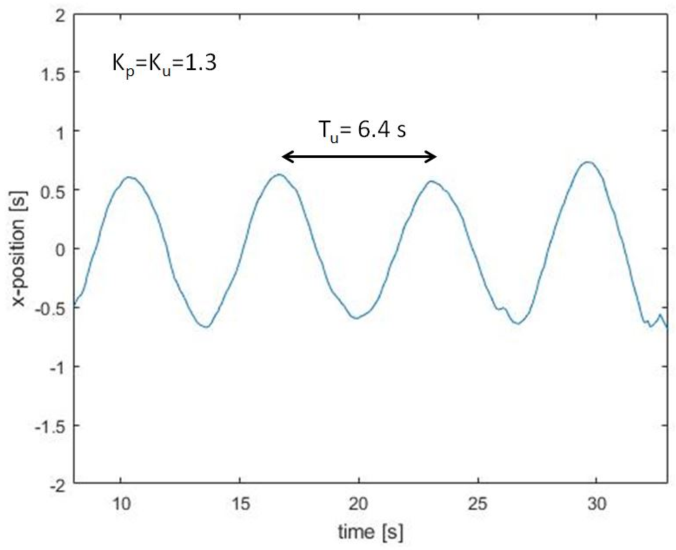


Figure 5.3: Position control of the drone with proportional control (K_p=K_u) in x-direction

$$K_{p,x} = 0.8K_{u,x} = 1.04 \tag{5.25}$$

$$T_{d,x} = \frac{T_{u,x}}{8} = 0.8 \tag{5.26}$$

The parameters of the PID controllers for the y- and z-direction were obtained analogously.

5.5 Evaluation of the control

Unfortunately, this method resulted in large overshoots. Fine-tuning of the parameters and other autotuning methods like the Åström-Hägglund tuning method had similar or worse results.

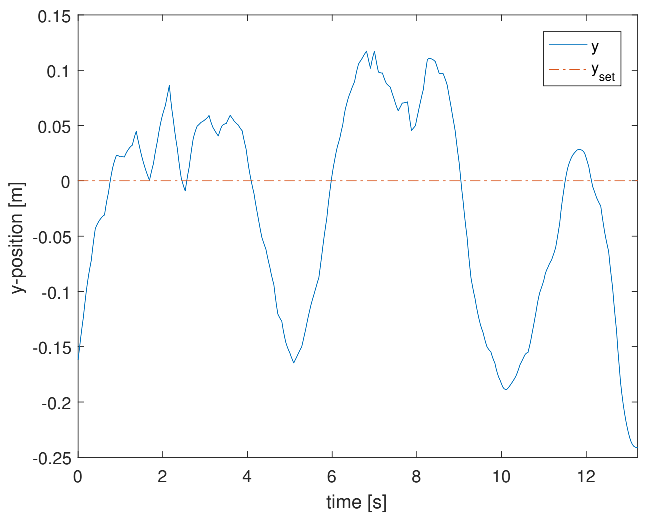


Figure 5.4: y-position control while applying constant velocity in x-direction

This was remarkable, as the same method gave good results when only the IMU was used for localization. This can be explained by the way that the Pozyx is fed. Due to the weight restrictions of the drone, it was not possible to feed the Pozyx with its own battery. That is why the Pozyx is fed directly from the battery of the drone. The power wires that connect the battery from the drone and the Pozyx cause a gap between the hull and the drone itself as can be seen in figure 5.5. This does not only affect the dynamics of the drone strongly, but it also causes imbalance of the drone when it just hovers. This makes the drone very unstable. The drone does not have the capability to fully compensate for this instability fast enough and consequently it is not able to avoid high overshoot.

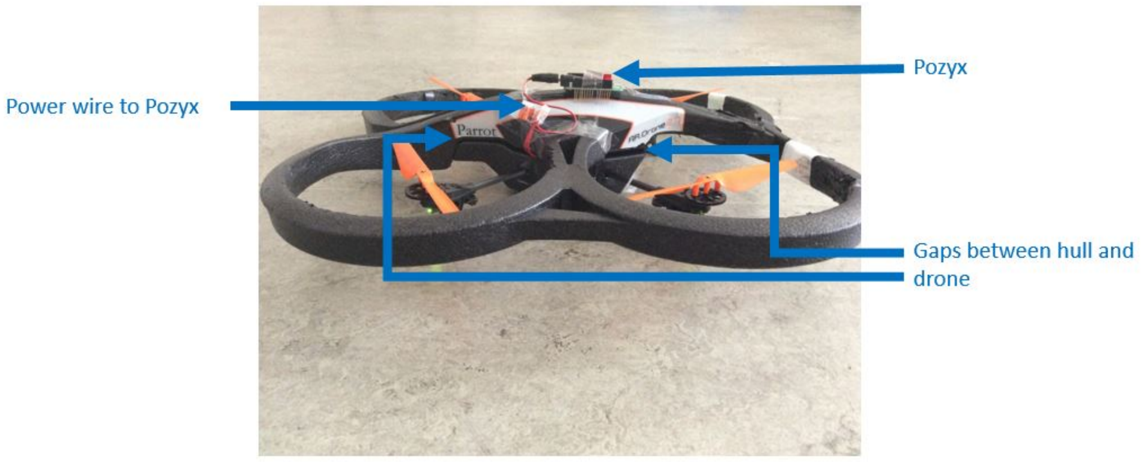


Figure 5.5: Gaps between hull and drone resulting from the power wires of the Pozyx

Chapter 6

Obstacle Detection Algorithm

Obstacle detection is one of the key functionalities of a fully autonomous drone. As a proper detection of obstacles has many requirements, it is also one of the biggest challenges in the development process. In this chapter, a subdivision is made between the conceptual implementation and how it should be implemented in reality. For the conceptual implementation, a drone is used that can only carry very limited extra weight. In practice, this means that the drone cannot be equipped with extra sensors, which limits the possible obstacle detection hardware to the onboard sensors. The remark needs to be made that the focus of the obstacle detection is on the detection of obstacles on the ground, but some of the insights can be extended to obstacle detection for the path planning.

6.1 Case I: Static obstacles

For the detection of static obstacles a solution based on distance measurements is proposed. The algorithm uses the difference in absolute height (the distance to the ground level) and relative height (the distance to the nearest obstacle) to detect an obstacle. A graphical representation of the heights can be found in figure 6.1.

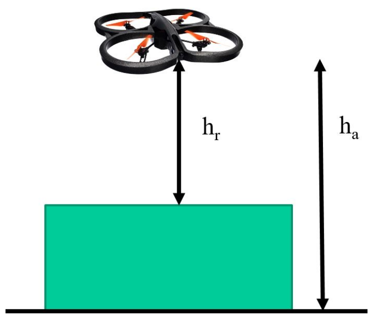


Figure 6.1: Absolute and relative height

6.1.1 Concept of static obstacle detection

For the measurement of the distance to the nearest obstacle (relative height), an ultrasonic sensor is used as the drone is already equipped with one. This sensor has some downsides concerning accuracy, but the accuracy is sufficient to show the concept. To measure the absolute height, the drone uses the z-position resulting from the localization algorithm. When the difference between the absolute height and the relative height exceeds a certain threshold value, it is assumed that an obstacle is present on that position.

Correction for roll and pitch

As the drone has a certain roll and pitch, these parameters should be taken into account in order to calculate the position of the measuring point and the relative height.

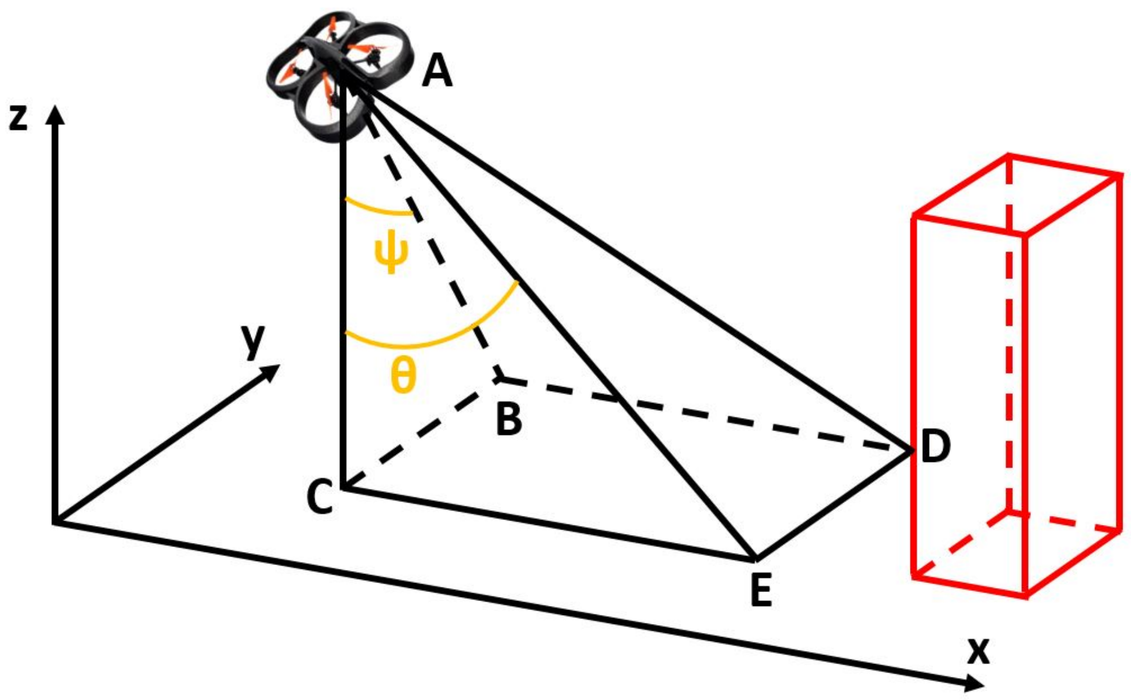


Figure 6.2: Position and relative height correction due to pitch and roll

In figure 6.2 a graphical representation can be found where A (x_A,y_A,z_A) is the position of the drone, |AD| is the distance measured by the ultrasonic sensor, ψ is the roll, θ is the pitch and the red object is an obstacle. From this information we can derive the maximum relative height at position D.

$$|AE| = |AD| \cdot cos(\psi) \tag{6.1}$$

$$|AB| = |AD| \cdot cos(\theta) \tag{6.2}$$

$$|AC| = |AD| \cdot cos(\psi) \cdot cos(\theta) \tag{6.3}$$

$$x_D = x_A + sin(\theta) \cdot |AE| = x_A + sin(\theta) \cdot |AD| \cdot cos(\psi) \tag{6.4}$$

$$y_D = y_A + sin(\psi) \cdot |AB| = y_A + sin(\psi) \cdot |AD| \cdot cos(\theta) \tag{6.5}$$

$$h_{r,max} = |AC| = |AD| \cdot cos(\psi) \cdot cos(\theta) \tag{6.6}$$

If $h_a - h_{r,max} > h_{thresh}$, an obstacle is present. If $h_a - h_{r,max} < h_{thresh}$, no conclusions can be drawn. The derivations above assume perfect measurements and allow high pitch and roll. In reality the measurement of the ultrasonic sensor is noisy and the pitch and roll have low values. In figure 6.3, the measuring point with and without correction can be found for a random trajectory of the drone above an obstacle. In figure 6.4, the relative height with and without correction can be found for the same trajectory. The relative heights almost coincide, while the measuring points with or without correction differ up to a few centimeters.

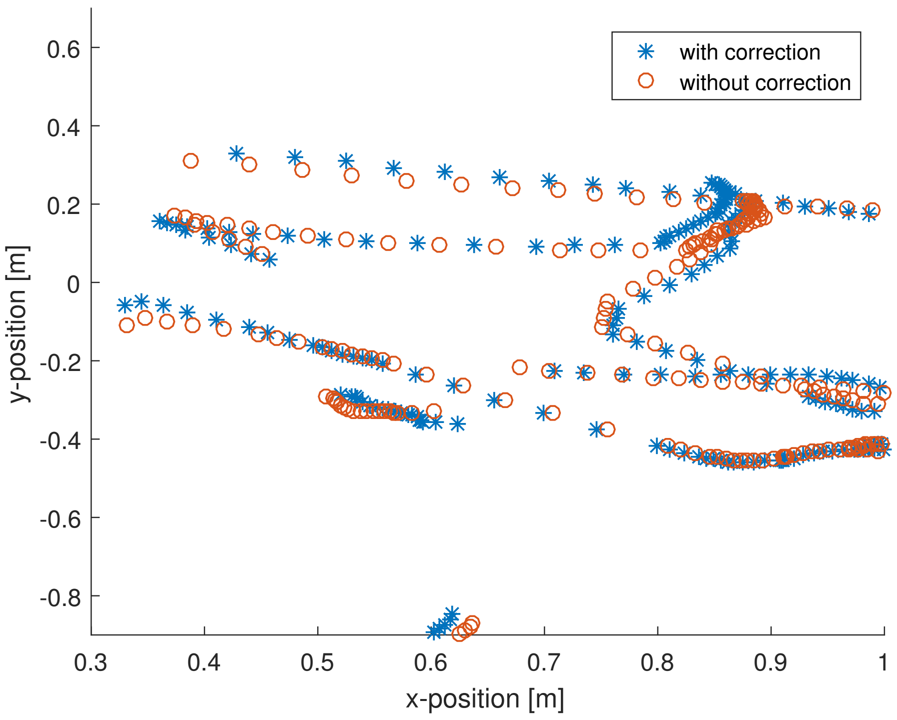


Figure 6.3: Position correction due to tilt of the drone

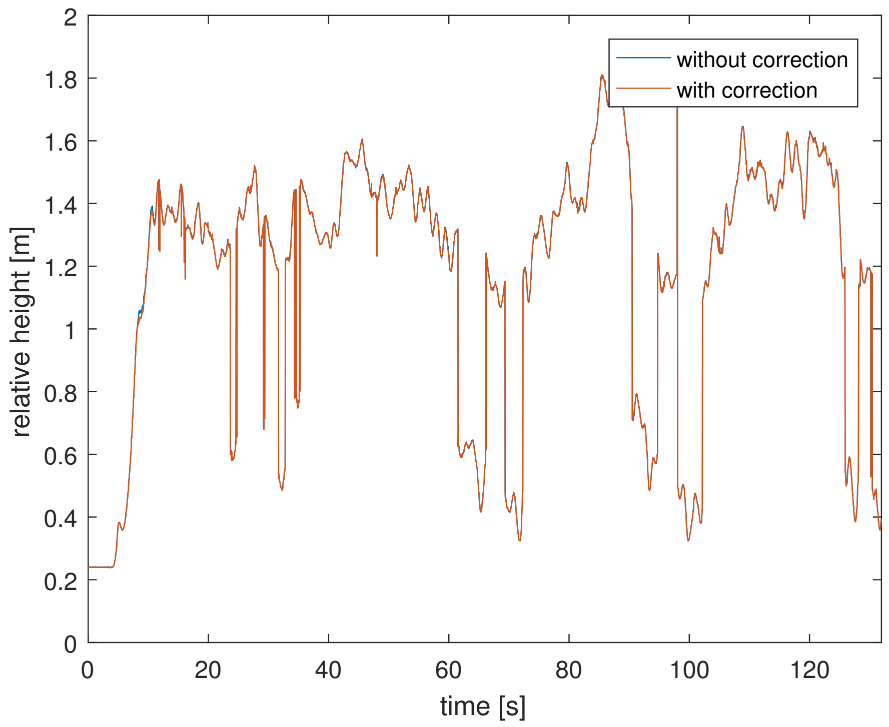


Figure 6.4: Relative height correction due to tilt of the drone

Mapping

To map the ground, obstacles are assumed with a rectangular shape and a certain length and width. Depending on the minimum length and width of the obstacles, the drone follows a certain trajectory. The distance between consecutive desired measuring points (D_{mp}) needs to be smaller or equal to $\frac{min(length_{obst,i}, width_{obst,i})}{2}$.

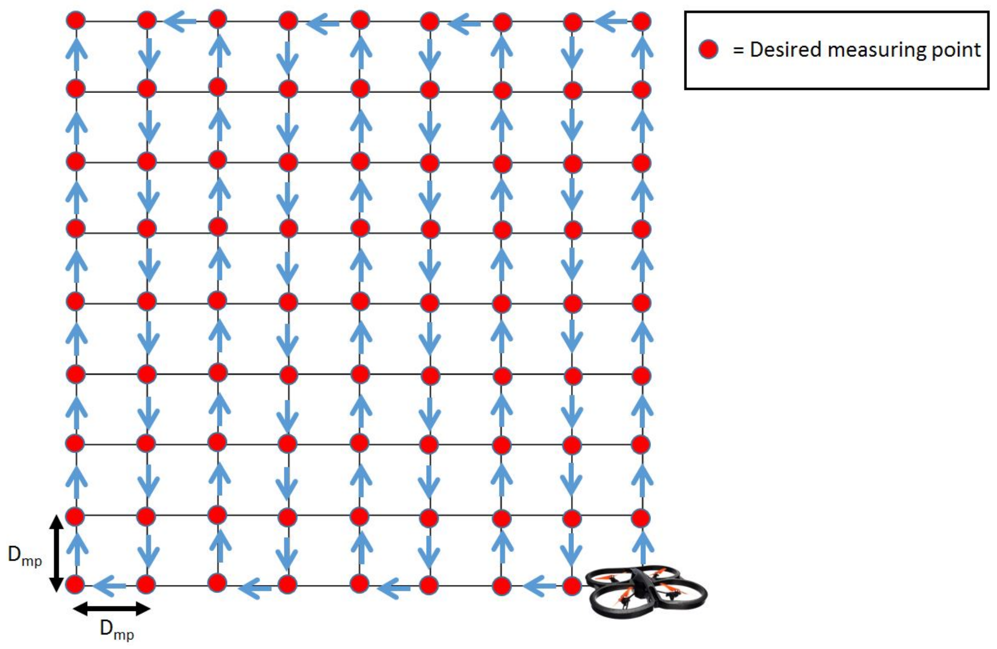


Figure 6.5: Measuring path

For each measuring point a deviation of D_{mp} is allowed in x- and y-direction. From the moment the drone has a measurement in the close neighborhood of the measuring point, the drone advances to the next measuring point.

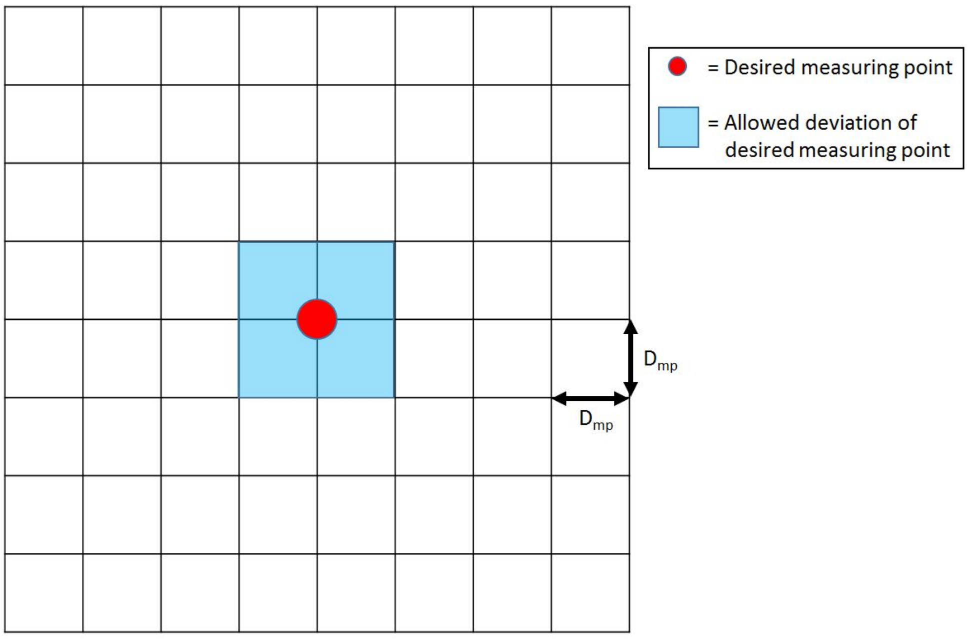

● = Desired measuring point
= Allowed deviation of desired measuring point
D_{mp}
D_{mp}

Figure 6.6: Allowed deviation measuring point

While the drone is flying, it constantly checks if an obstacle is present. When an obstacle is detected, the closest discrete point to the measuring point is marked as “obstacle detected”.

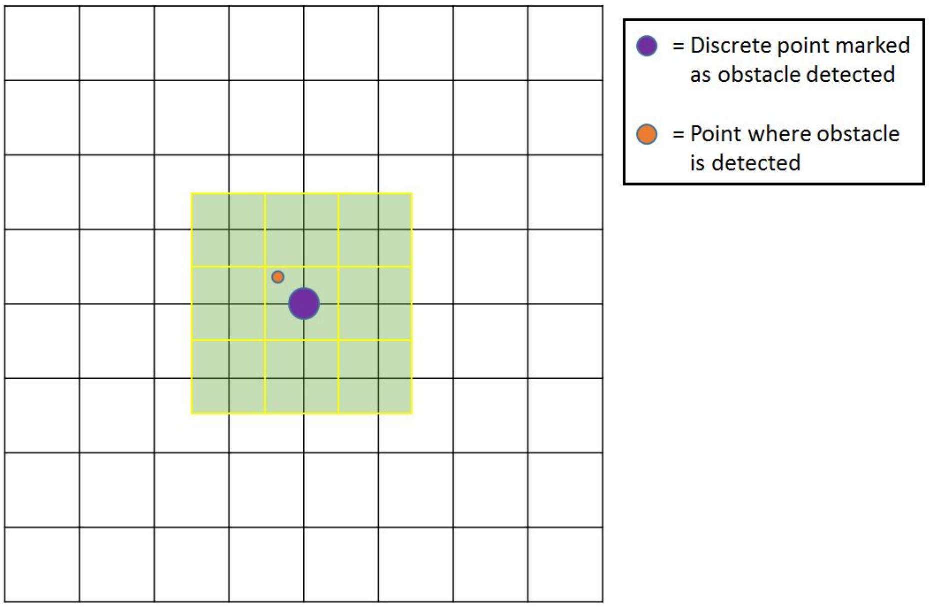

● = Discrete point marked as obstacle detected
● = Point where obstacle is detected

Figure 6.7: Discretization measuring point

Figure 6.8 shows the outcome of the mapping of an environment with one obstacle in it. The result of the mapping is a binary matrix.

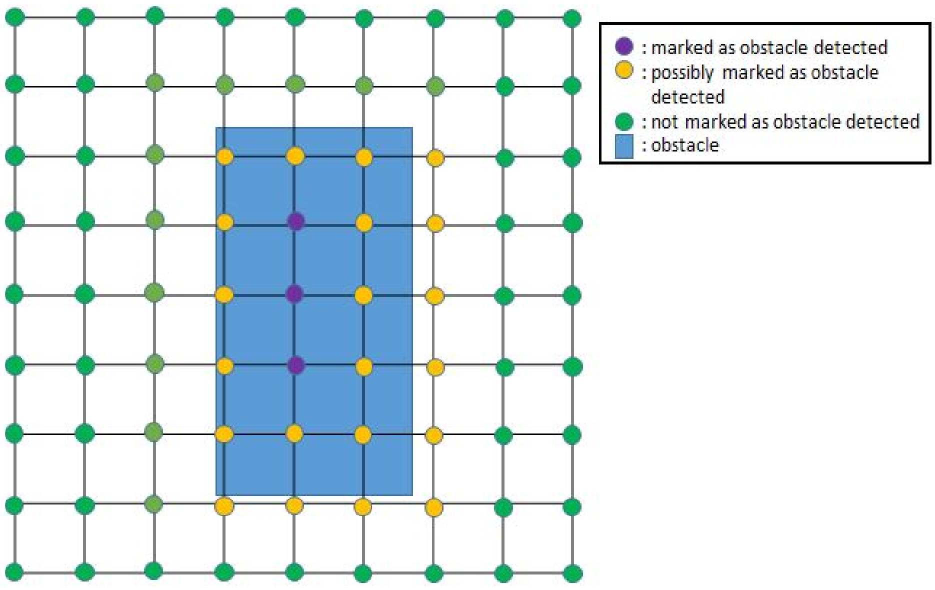

● : marked as obstacle detected
● : possibly marked as obstacle detected
● : not marked as obstacle detected
▨ : obstacle

Figure 6.8: Result of mapping

To be sure that the whole obstacle is covered, each point marked as “obstacle detected” should be extended by $2 \cdot D_{mp}$ in every direction. The result in the worst-case scenario (when only the purple circles were detected) is given in figure 6.9.

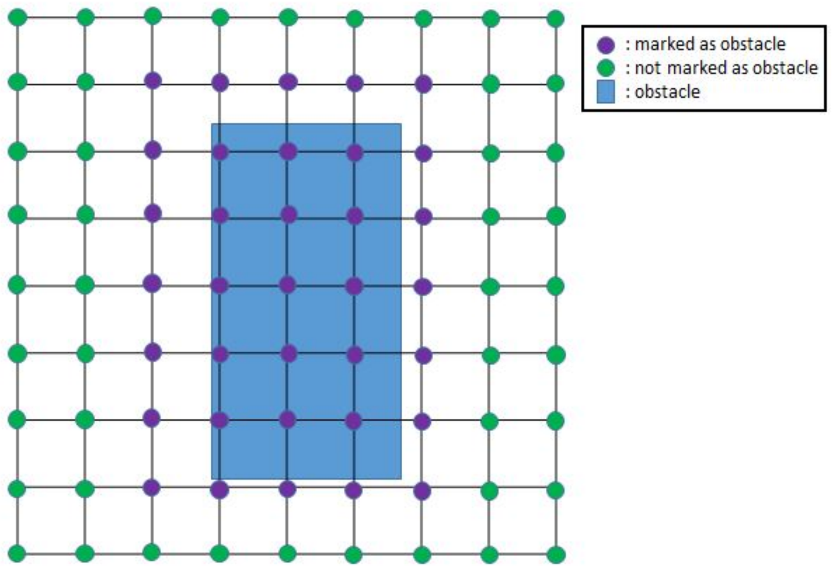


Figure 6.9: Result after processing of the map in worst-case scenario

Now the complete area of the obstacle is certainly marked as “obstacle”. In case that in the original mapping in figure 6.8 more points were marked as “obstacle detected”, the area that is marked as “obstacle” (after the processing of the map) can be much larger than the real obstacle. If this is a problem and a higher accuracy is desired, a lower value of D_{mp} should be chosen.

As the environment outside the mapped area is unknown, everything outside the mapped area needs to be marked as “obstacle”. This means that the obstacle map always needs to consist of a border of $2 \cdot D_{mp}$.

6.1.2 Static obstacle detection in real-world applications

In reality the goal is to gather information from the environment under the drone when it is hovering at a fixed position.

Determination of the relative height

For accuracy reasons it is recommended to use laser scanners to measure distances instead of ultrasonic sensors. As can be seen in figure 6.10, the ultrasonic wave propagates in a conical way. Due to this problem, it is not possible anymore from a certain height to map the ground accurately with an ultrasonic sensor.

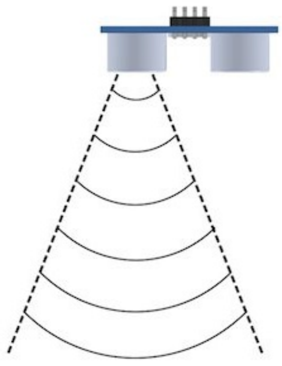


Figure 6.10: Ultrasonic sensor

Determine absolute height

In most indoor environments, the ground level has a fixed height difference with respect to the sea level. In this way it is sufficient to measure the absolute height with respect to a fixed reference. Outdoors the difference between the height of the ground level and the height of the sea level is not a fixed value anymore. In order to find the absolute height h_a, we need to measure the height of the drone with respect to the sea level (h_{drone}) and the elevation of the ground (h_{road}). A graphical representation can be found in figure 6.11.

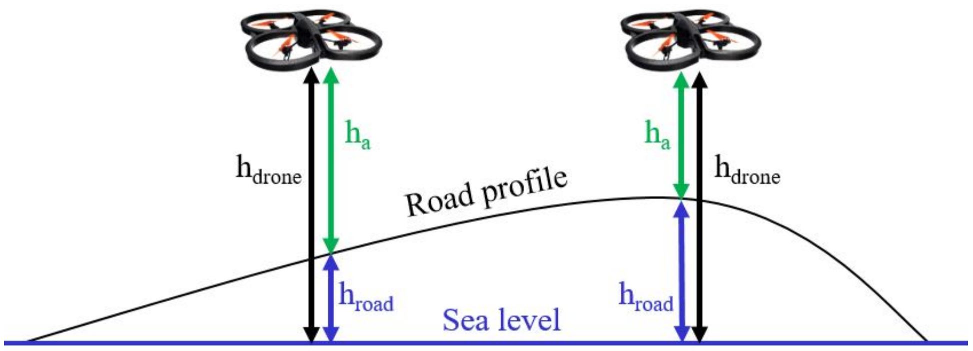


Figure 6.11: Absolute height in reality

To measure the height of the drone, the information from a GPS can be fused with the information from a pressure sensor. To measure the elevation of the ground, a database should be used that returns the ground elevation for a given position (latitude and longitude) of the drone. The Google Maps Elevation API[1] is an example of such a database. It provides elevation data

[1]https://developers.google.com/maps/documentation/elevation/start

for all locations on the surface of the earth. This elevation data contains an estimation of the elevation at a specific point and the maximum distance between data points from which the elevation was interpolated. The position of the drone can be estimated outdoors in a similar way as in chapter 4, of course a GPS is used instead of the Pozyx.

Mapping at fixed position

To solve the problem that the drone needs to fly a certain path to have an idea of the ground level, the laser scanner can use deflecting mirrors, which enables them to achieve wide fields of vision. In order to have a 360° visibility, the laser scanner should be able to rotate around its own axis. The laser scanner can have its own motor or the laser scanner can rotate with the propeller of the drone. As only static obstacles are assumed it is also possible that the drone itself just rotates around its own axis, this is a cheaper solution.

Existing technology

A lot of sensors with different specifications are already commercially available for UAVs. Most laser scanners are based on LIDAR and use the methods described above, whether or not fused with other sensors.

6.2 Case II: Moving obstacles

To recognize unknown moving obstacles, more advanced sensors are needed than the ones that are currently installed on the Parrot AR.Drone 2.0. As mentioned earlier, it is not possible to install extra sensors due to weight restrictions. That is why the moving obstacles are marked so that they can be detected with the bottom camera. Although this method seems artificial, it strongly relates to possible methods for the real-world application as will become clear in the following subsections.

6.2.1 Algorithm implemented in the moving obstacles

In the experiments, Lego Mindstorms EV3 were used to represent the moving obstacles. In the Mindstorms themselves there is also obstacle avoidance implemented. The bot uses an ultrasonic sensor to detect the obstacles. The ultrasonic sensor measures the distance to the closest obstacle.

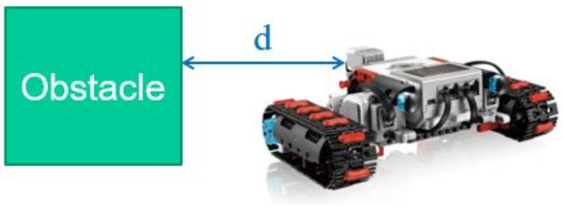


Figure 6.12: Distance measurement with ultrasonic sensor of Lego Mindstorm EV3

The algorithm works according to the flowchart displayed in figure 6.13. A rather high value is chosen for the distance from when the bot starts to turn around its axis, this is to buffer for the finite dimensions of the bot. To avoid constantly switching between turning and going straight forward, hysteresis is added to the system.

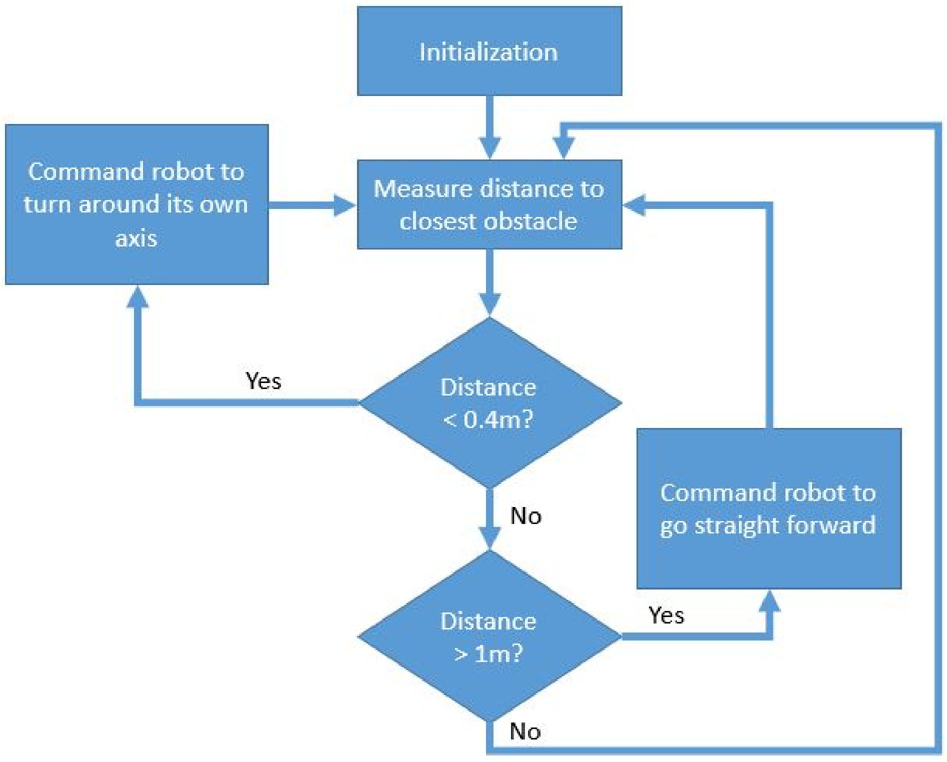

Initialization
Command robot to turn around its own axis
Measure distance to closest obstacle
Yes
Distance < 0.4m?
No
Command robot to go straight forward
Yes
Distance > 1m?
No

Figure 6.13: Flowchart of obstacle avoidance algorithm that is implemented in the Lego Mindstorms

6.2.2 Concept of dynamic obstacle detection

To show the concept, the moving obstacles are marked with colored paper. The obstacles are then filtered from the figure based on their color. From this, the position of the obstacles can be determined.

Color space transformation

The best known color space is RGB, it represents a color by combining a certain amount of the red color, the green color and the blue color.

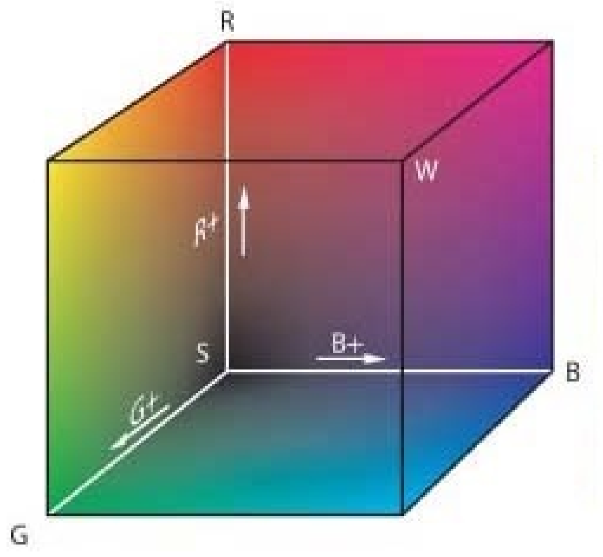


Figure 6.14: RGB color space[2]

For image processing purposes, it is useful that the color is separated from the intensity. It results in more robustness to lighting changes. That is why the HSV (Hue, Saturation and Value) color space will be used for the color detection.

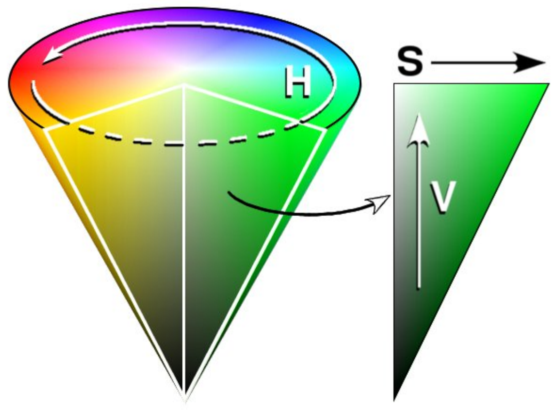


Figure 6.15: HSV color space[3]

[2]Figure from Wikimedia Commons[40]
[3]Figure from Wikimedia Commons[39]

Color detection

In order to be able to detect the position of the colored paper, each pixel that has a certain HSV value should be converted to a binary value that indicates if the obstacle (colored paper) is present in that particular pixel. To be able to do this conversion, a range of hue, saturation and value needs to be specified that corresponds to the colored paper. In figure 6.16, a picture of a red paper can be found that is taken by the bottom camera of the drone. The green rectangle shows the information that needs to be extracted from the picture. In figure 6.17, the specified range of HSV values and the converted picture to binary values can be found.

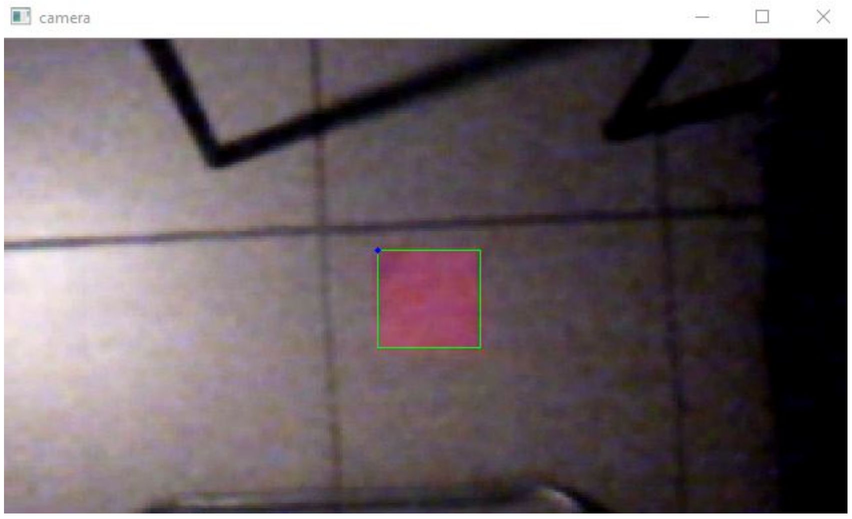
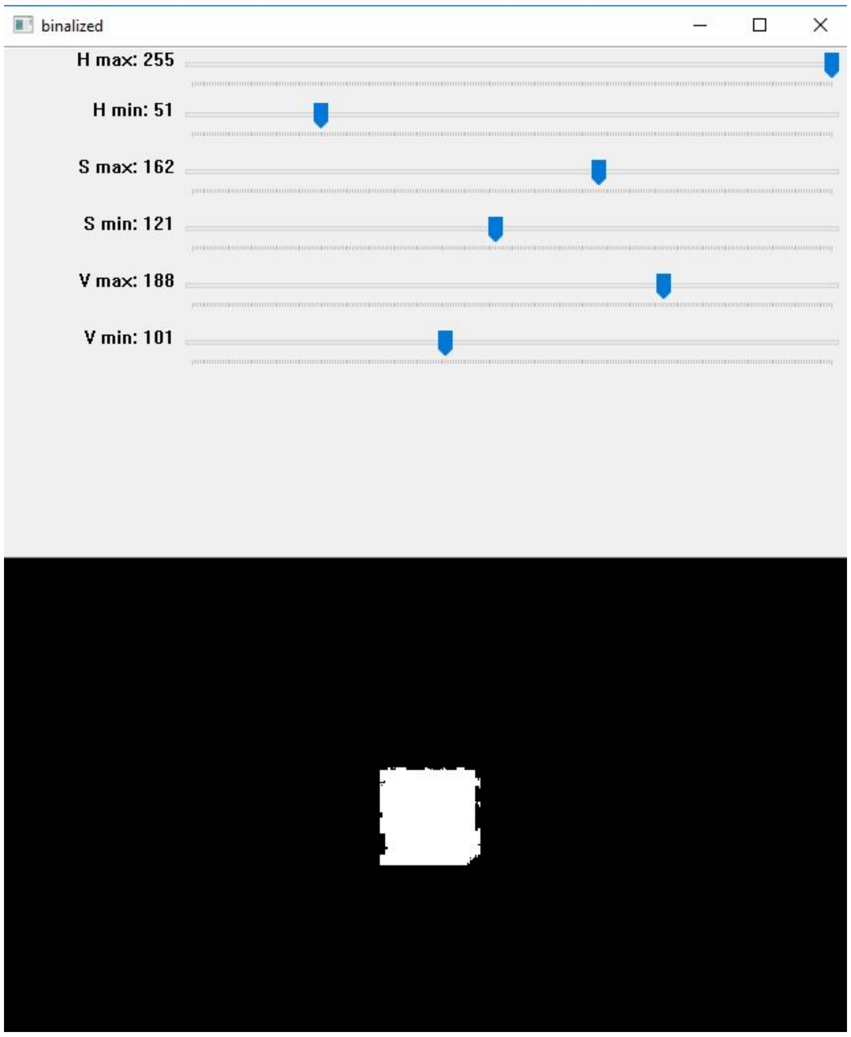


Figure 6.16: Picture of red paper taken by bottom camera

To obtain a rectangle that includes the object as in figure 6.16, the standard contour tracing function of OpenCV is used. This function implements the algorithm described in “Topological structural analysis of digitized binary images by border following”[37].


binalized

H max: 255

H min: 51

S max: 162

S min: 121

V max: 188

V min: 101

Figure 6.17: Converted picture from HSV to binary

Conversion from pixel coordinates to real-world coordinates

From the rectangle resulting from the contour detection, it is possible to extract the pixel coordinates of the obstacle. These coordinates still need to be converted to real-world coordinates in order to have an estimation of the position and the velocity of the obstacle.

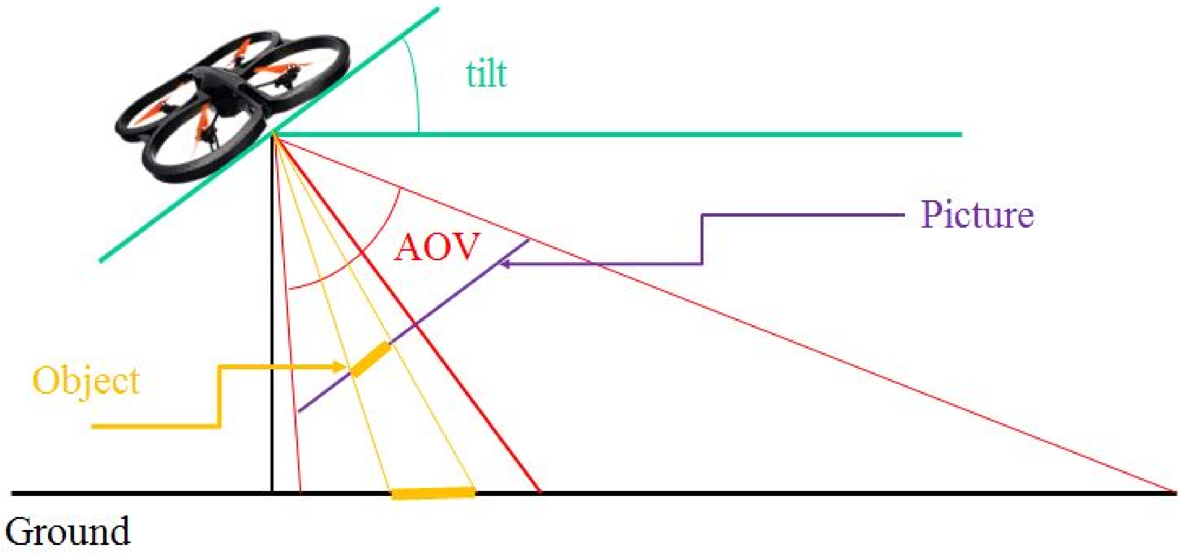


Figure 6.18: Relationship between an object in pixel coordinates and in real-world coordinates in 1D

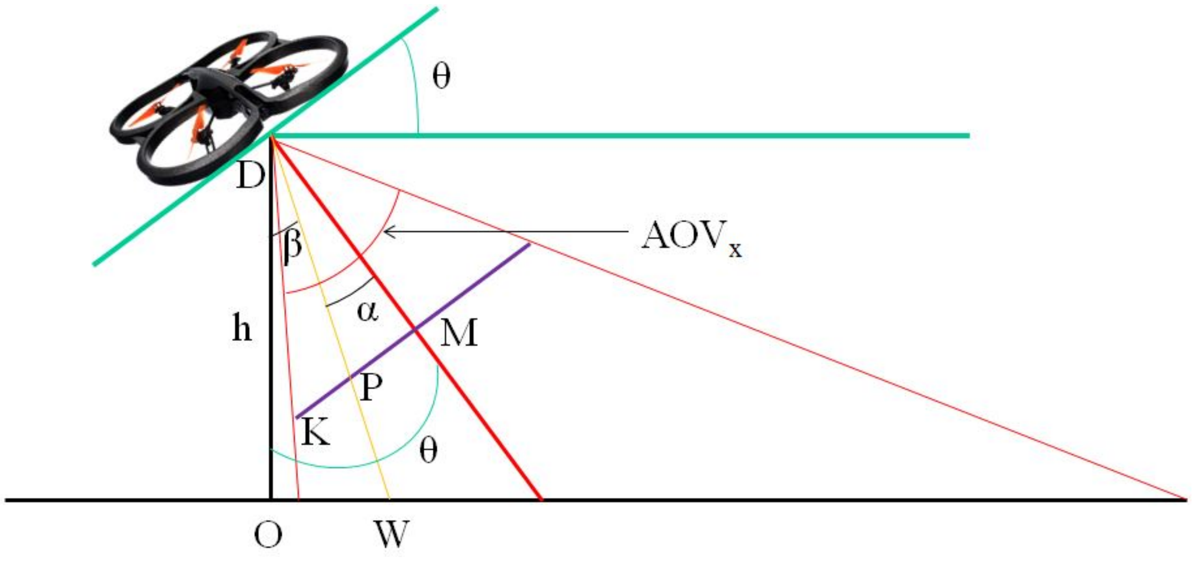


Figure 6.19: Conversion from point in pixel coordinates to real-world coordinates in 1D

$|PM|$ is equal to $|X_P - X_M|$ with X_M the x-coordinate of the pixel that is in the middle of the picture ($=\frac{total\ amount\ of\ pixels\ in\ x-direction}{2}$) and X_P the x-coordinate of the pixel of which the position in world coordinates needs to be known.

$$|DM| = \frac{|KM|}{\tan(AOV_x/2)} \tag{6.7}$$

$$\alpha = \arctan(\frac{|PM|}{|DM|}) \tag{6.8}$$

β is then equal to $\theta \pm \alpha$, the sign depends whether $X_P > X_M$ or $X_P < X_M$ that is why α' is introduced.

$$\alpha' = \arctan(\frac{X_P - X_M}{|DM|}) \tag{6.9}$$

Now the required sign of α is included in α' such that:

$$\beta = \theta + \alpha' \tag{6.10}$$

From which the x-coordinate in drone coordinates (with respect to the drone) results:

$$x_W = \tan(\beta) \cdot h \tag{6.11}$$

The absolute world x-coordinate of the object is then:

$$x_{obj} = x_{drone} + \cos(yaw) \cdot x_W - \sin(yaw) \cdot y_W \tag{6.12}$$

The derivation of the y-coordinate is completely analogous:

$$y_{obj} = y_{drone} + \cos(yaw) \cdot y_W + \sin(yaw) \cdot x_W \tag{6.13}$$

Estimation of the velocity of an obstacle

Since the position of the object can be calculated, the velocity can be estimated by taking the finite difference of the position.

$$v_{obj}(t_2) = \sqrt{\left(\frac{x_{obj}(t_2) - x_{obj}(t_1)}{t_2 - t_1}\right)^2 + \left(\frac{y_{obj}(t_2) - y_{obj}(t_1)}{t_2 - t_1}\right)^2} \tag{6.14}$$

In order to avoid high fluctuations of the velocity that are the result of noise, t_2-t_1 should be taken high enough.

6.2.3 Dynamic obstacle detection in real-world applications

Marking every obstacle that needs to be detected is impossible. That is why it is proposed to equip the drone with stereoscopic vision. Stereoscopic vision is receiving images from different points of view. First, the point of interest needs to be transformed to the corresponding pixel coordinate in each image. With this information, the 3D position of that point can be calculated using triangular geometry in a similar way as was done for the conceptual obstacle detection. Once a 3D image is constructed, the obstacles can be segmented and their position can be calculated. Since a 3D image needs to be rendered and interpreted each time, this method will be computationally expensive.

Parrot, also the manufacturer of the AR.Drone, recently launched an all-in one integrated kit named Parrot S.L.A.M. dunk[4] to help developers to create advanced navigation applications. It is equipped with stereoscopic vision and an ultrasonic sensor which allows the creation of a 3 dimensional point cloud. In addition to the hardware, some software is included. It already offers 3D mapping and simultaneous localization (SLAM) using the powerful on-board processor. The embedded computer is based on Ubuntu and is delivered with a ROS compatible SDK.

One of the biggest shortcomings of this method is the fact that it cannot be used at night in outdoor environments. This is where time-of-flight cameras can offer a solution. Time-of-flight cameras measure the time between sending an infrared signal and receiving it. Using the speed of light, the distance can be estimated. The technology is similar to the laser scanners described in subsection 6.1.2. The difference is that, in contrast to laser scanners that scan the environment point by point, time-of-flight cameras capture the entire scene with each light pulse which makes it more suitable for dynamic environments.

In addition to position and velocity measurements, it may be useful for an autonomous drone to have an idea about the type of obstacle. In case of the cardiac arrest application, it is useful to know if an obstacle that is running towards the drone is a human being who wants to use the AED as soon as possible or if it is a wild animal that can possible damage the drone. It is also useful that the drone knows above what type of ground he is hovering. Not only is it safer for a drone to land on sidewalks, parkings or emergency lanes than it is to land on highways, certain types of ground such as rivers and mud pools will certainly cause damage to the drone. Object recognition can be done by building a classifier through training. Training is done by walking through a lot of pictures of which it is known beforehand if the specific object (e.g. a human being) is in the picture or not. The best results are currently achieved with obstacle detection based on deep learning. This technology has already been commercially implemented in advanced driver-assistance systems in cars. A well known company that produces such systems is Mobileye[5]. Mobileye devices support, inter alia, pedestrian, bicycle, motorcycle, vehicle, traffic sign and lane marking recognition.

[4]https://www.parrot.com/nl/en/business-solutions/parrot-slamdunk#parrot-slamdunk
[5]https://www.mobileye.com/

Chapter 7

Path Planning Methods

In order to get towards the target location, some path planning (with obstacle avoidance) needs to be done. A lot of different paths can be generated between two positions. An optimal path can be determined based on preferences. If speed is important, a short path should be chosen. Whereas a longer path should be chosen when safety (larger distance to obstacles) is more important.
The acquired insights of this chapter are primarily deduced from "Evaluation of path planning algorithms for a quadcopter during indoor flight"[6] written by Diericx, a former student of Ghent University. The evaluation at the end of this chapter is the result of simulations where the focus is on finding a proper path planning algorithm. The detection of the obstacles is not addressed here.

7.1 Method I: Graph-based path planning

To reduce the computation time, the environment can be discretized. Grid graphs, Voronoi diagrams and visibility graphs are the most commonly used graphs.

7.1.1 Grid graph

The environment is divided into cells. If an object is detected in a cell, the whole cell is marked as inaccessible to the drone. As the drone has finite dimensions, the radius of the drone needs to be taken into account. An easy way to do it is by just adding the radius of the drone to every obstacle.

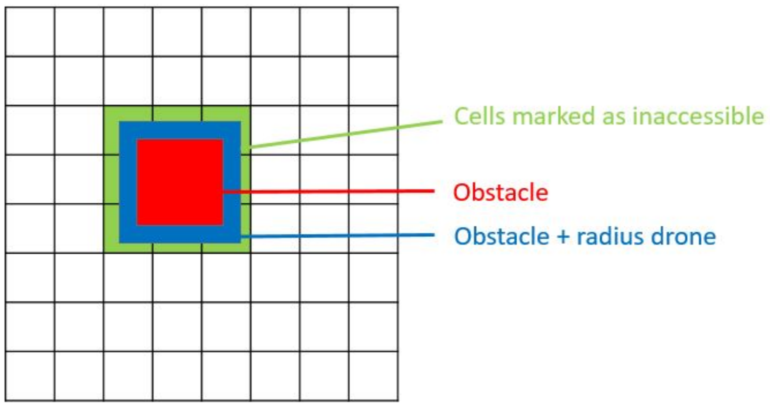


Figure 7.1: Grid graph

Resolution

The choice of the resolution is a trade-off between a detailed map and a low computation time. A high resolution results in an accurate map of the environment and offers a larger amount of possible paths towards the location of the victim. However, a higher resolution also results in a higher computation time as the computation time rises exponential with the resolution.

Hierarchical mapping

To profit from the benefits of a high resolution and the ones of a low resolution, hierarchical mapping can be used. The critical sections where obstacles are detected are then subdivided into a higher resolution. In this way it is possible to specify a better path with a limited increase of the computation time.

7.1.2 Voronoi diagram

A Voronoi diagram is a set of lines and points that are equidistant to the two closest obstacles. The resulting path is as far as possible away from the obstacles.

7.1.3 Visibility graph

In a visibility graph all corners of obstacles are connected when the connection does not intersect with another obstacle. The obstacles need to be extended again with the radius of the drone, to find proper paths. It is possible to find the absolute shortest path when a shortest path algorithm is applied. This path can then be used to compare other algorithms.

7.1.4 Cost function

To find an optimal path between two points, certain costs need to be specified. The cost needs to be calculated to go from one node to another one on the graph. This cost can consist of:

- **Distance cost**
  This cost penalizes longer paths.
- **Height cost**
  Depending on the height of an obstacle it can be better to avoid collision by flying over the obstacle or by flying around it.
- **Near obstacle cost**
  Paths that are close to an obstacle can be faster but less safe than other paths.
- **Turn cost**
  A smooth path can result in a faster path than a shorter path.
- **Uncertainty cost**
  In reality no complete mapping is available at each time instant. This cost penalizes paths that are going through an area that is not mapped yet.

These costs are weighed by preference, the optimal path is then the path that has the lowest total cost.

7.1.5 Path planning algorithms

The algorithms described below are based on Dijkstra's algorithm. They primarily focus on the optimization of the computational cost.

A*[27]

This algorithm converges faster to a solution by making use of a heuristic. The weight attributed to the heuristic determines whether a low computation time or a high certainty of the optimal path is preferred.

D* Lite[38]

D*Lite is an optimized version of D*. It primarily tackles the problem of dynamic environments. If new obstacles frequently appear, it is not recommended to recalculate the path each time. That is why D*Lite and D* uses the previously calculated graph to calculate the path faster.

Anytime Dynamic A*[22]

Anytime algorithms focus on finding a (possibly suboptimal) solution very quickly. It improves this solution until the planning time runs out. Dynamic algorithms are efficient replanning algorithms that cope with incomplete information and changing environments. The Anytime Dynamic A* algorithm is a combination of the algorithms that are described above.

Field D*[8]

In the previously described algorithms, the heading of the agent is constrained to discrete steps of 45 degrees from 0 to 360 degree. The Field D* algorithm applies linear interpolation to the path costs to find a more optimal path. In this way paths are produced with a range of continuous headings.

7.2 Method II: Potential fields

Potential fields is another approach for path planning. It applies repelling forces from obstacles and attractive forces towards the target location. It results in a more natural and smoother movement of the agent.

7.2.1 Forces

The forces consists of an attractant force to the target location, a repellent force from the obstacle and an extra attractant force to the target location. The last force is introduced because of the danger that the repellent force is too strong when the target location is close to an obstacle. Without the extra force it is possible that the agent is stuck into a local minimum.

$$U_{target} = W_{target} \cdot d_{target} \tag{7.1}$$

$$U_{obst} = \sum_{i=1}^{n} \frac{W_{obst,i}}{d_{obst,i}^2} \tag{7.2}$$

$$U_{extra} = -\frac{W_{extra}}{d_{target}^2} \tag{7.3}$$

With U the potentials, W the weights and d the distance from the agent to the different locations.

7.2.2 Path planning algorithm

To find the optimal heading of the agent, different implementations are possible. The problem can be solved by applying local neighborhood search. The search can be done by taking some samples around the agent or by taking fewer samples and applying interpolation in order to lower the computation time. The next position is then the neighbor with the lowest value. Another way to find the optimal heading is by using gradient descent.

7.2.3 Local minima

The danger exists that in complex environments a lot of local minima appear what leads to possible problems to find the global minimum (target location). This can be solved by adding penalties to the current position, although there is a danger that the extra force causes a collision with an obstacle. Another possible solution is ant colony optimization. In this solution, the results are analyzed when the agent would go in different directions while only considering the repelling forces of the obstacles. The idea is that some starting directions will lead to a location that is closer to the target location. When the attracting forces are now taken into account again, the agent should be able to reach the target location.

7.3 Evaluation and Comparison Study

The results of experiments done by Diericx in"Evaluation of path planning algorithms for a quadcopter during indoor flight"[6] are discussed in this section. Grid graph, visibility graph and potential field are compared. As the different grid graph algorithms have similar outcomes, they are not evaluated separately.

7.3.1 Criteria

Below is a list of the different criteria to compare the different methods.

- Did the agent reach the target location?
- Are there any collisions with obstacles?
- Distance of path compared to the the shortest path
- Amount of turning
- Processing time

7.3.2 Environments

Different environments are considered.

- Environment 1
  This constrained environment includes some easily avoidable obstacles.
- Environment 2
  This is again a constrained environment but some complexity is added by putting the target location behind an obstacle. In this way it is harder to reach the target location.
- Environment 3
  This constrained environment has at first sight two possible paths to reach the target location. The shortest path contains an obstacle which was not included in the original map.

7.3.3 Results

The results of the simulations are listed in a table for each environment.

Environment 1

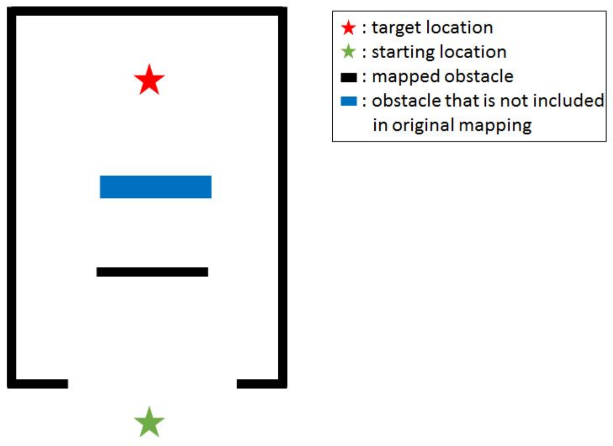


Figure 7.2: Graphical representation of environment 1

Table 7.1: Results environment 1

	Grid graph	Visibility graph	Potential fields
Target location reached?	yes	yes	yes
Collisions with obstacles?	no	no	yes
Distance compared to shortest path	+ 5.6 %	+ 0.0 %	+ 7.0 %
Amount of turning	123°	69°	214°
Processing time	15 ms	2 ms	0 ms

Environment 2

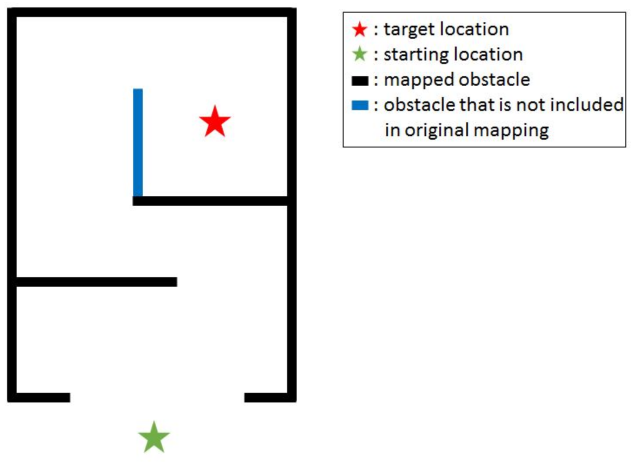


Figure 7.3: Graphical representation of environment 2

Table 7.2: Results environment 2

	Grid graph	Visibility graph	Potential fields
Target location reached?	yes	yes	no
Collisions with obstacles?	no	no	no
Distance compared to shortest path	+ 18.3 %	+ 0.1 %	/
Amount of turning	404°	309°	/
Processing time	15 ms	3 ms	/

Environment 3

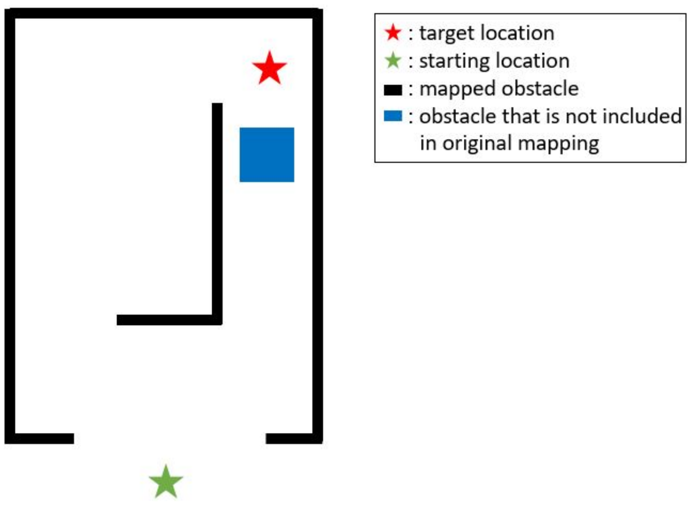


Figure 7.4: Graphical representation of environment 3

Table 7.3: Results environment 3

	Grid graph	Visibility graph	Potential fields
Target location reached?	yes	yes	no
Collisions with obstacles?	no	no	yes
Distance compared to shortest path	+ 81.5 %	+ 45.3 %	/
Amount of turning	627°	457°	/
Processing time	13 ms	3 ms	/

In the simulations, potential fields seems to be far less robust than graph-based algorithms. Although some solutions are proposed, the potential fields method tends to get stuck in local minima in many cases. The difficulty for potential fields lies in finding an appropriate way to assign weights that are suitable for every situation. Grid graph provides a more robust solution, the downside is the trade-off between accuracy and computation time. However, this can be partly solved by using a more efficient algorithm. For the tested environments, the visibility graph offers the best results.

7.3.4 Conclusion

When interpreting the results, one should take into account that these results are obtained for strongly constrained environments. It makes sense that a more natural approach as potential fields performs worse in such environments. When looking at the real-world application, the environment is far less constrained during flight. To thoroughly assess the outdoor performance of the different path planning methods, research should be conducted in less constrained environments. In addition, many dynamic obstacles such as birds and UAVs can be present. This is a very important factor to take into account. Built on the path planning algorithms for static obstacles, path planning algorithms for dynamic obstacles are developed [24][41][33]. By taking into account the less constrained environment and the presence of dynamic obstacles, more accurate results for the real-world application regarding path planning will be obtained.

Chapter 8

Landing algorithm

For the landing algorithm, only static obstacles are considered. The obstacle detection already provides the places were it is impossible to land due to possible obstacles, but the drone cannot land just next to an obstacle due to the finite dimensions of the drone. That is why the area that is marked as “obstacle” should be extended by the (rounded up) radius of the drone. For the following figures the radius of the drone is assumed equal to D_{mp}.

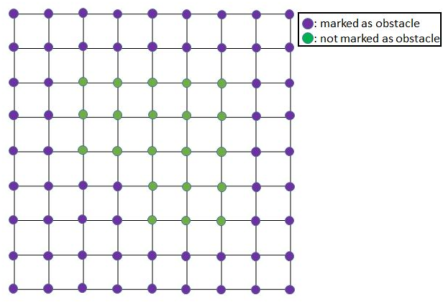


Figure 8.1: Obstacle map

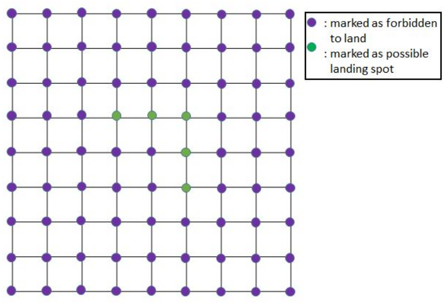

● : marked as forbidden to land
● : marked as possible landing spot

Figure 8.2: Landing map

The remark should be made that in figure 8.1 an infinite small drone could also land on the purple dots that form the border of the green ones as the purple dots are a limit case of the presence of an obstacle. This means that in theory the green dots in figure 8.2 could be extended by D_{mp}. It has been chosen to not do this and use this D_{mp} as a safety buffer.

As the landing map is a binary matrix, the optimal landing spot can quickly be found by checking for every possible landing point the distance to the target location. The optimal landing spot is than the one with the smallest distance to the victim’s location.

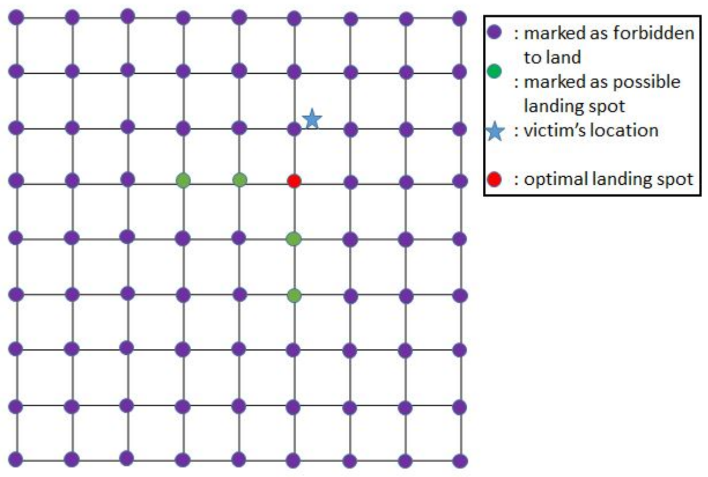

● : marked as forbidden to land
● : marked as possible landing spot
★ : victim's location
● : optimal landing spot

Figure 8.3: Optimal landing points for static obstacles

Chapter 9

Experimental Validation

In this chapter the results of the experiments are discussed.

9.1 Static obstacles

In the experiment, the methods are applied that are explained in previous chapters. A graphical representation of the environment where the experiment takes place, can be found in figure 9.1.

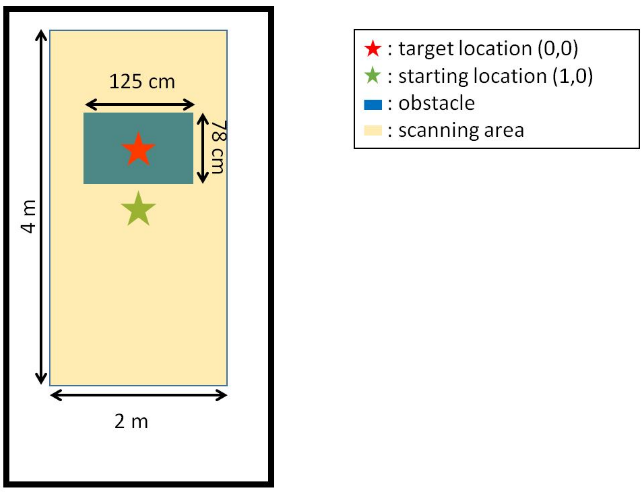


Figure 9.1: Graphical representation of environment where the experiment with static obstacles takes place

Obstacle detection

For the detection of static obstacles, the drone needs to follow a certain trajectory. A graphical representation of the path of the drone and the desired path can be found in figure 9.2. As mentioned earlier, the drone is suffering from a lot of overshoot.

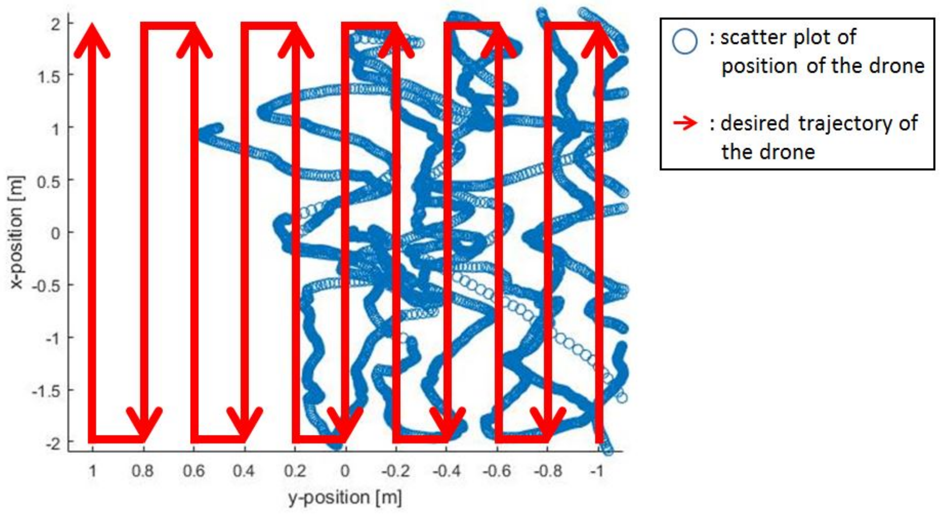


Figure 9.2: Desired and real path of the drone in experiment with static obstacles[1]

At each point the absolute height is compared to the relative height. When this value exceeds a certain threshold value (0.8 in the experiments), it is internally stored that an obstacle is present on that location.

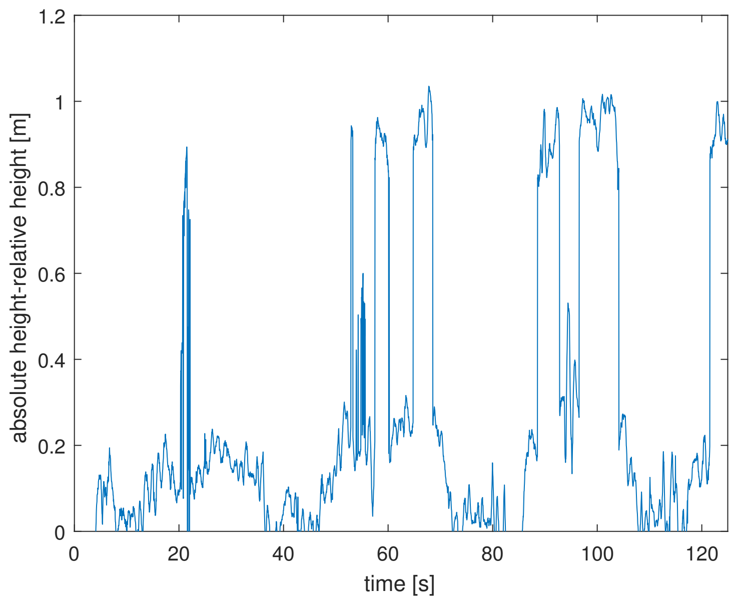


Figure 9.3: Absolute height - relative height in experiment with static obstacles

[1]It can be noted that the trajectory is not completely finished by the drone, this only due the amount of logged points that was limited to 4000.

Obstacle map

Using figure 9.3, the locations where obstacles are detected, can be extracted from figure 9.2.

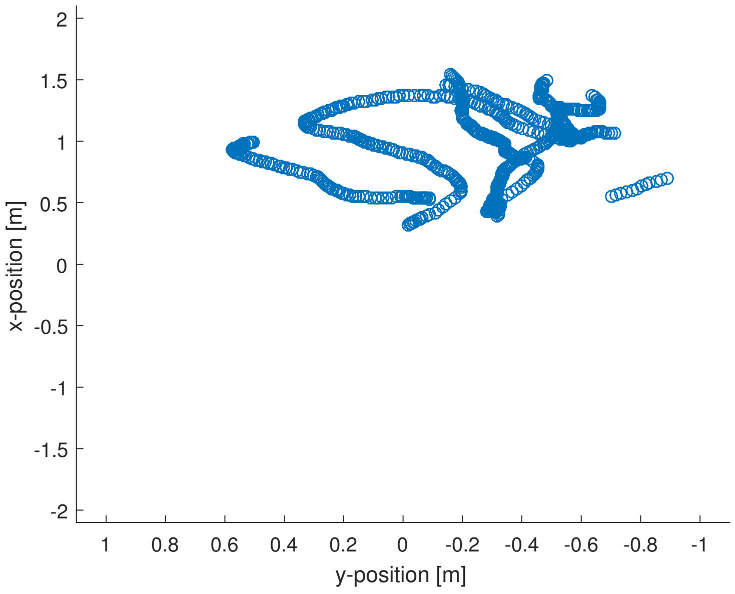


Figure 9.4: Locations where obstacles are detected in experiment with static obstacles

The remark needs to be made that the values from figure 9.4 have been obtained by post-processing of the logged values. When an obstacle is detected during the experiment, the “analog” position is internally immediately converted to the corresponding discrete position. The discrete positions are represented by a boolean matrix with $round(\frac{width\ of\ scanning\ area}{D_{mp}})+1$ columns and $round(\frac{length\ of\ scanning\ area}{D_{mp}})+1$ rows. In this way the whole room is mapped while only very few boolean values need to be saved. This method does not only reduce the necessary memory, but it also reduces the computation time drastically. The internally saved map of the obstacles can be found in figure 9.5.

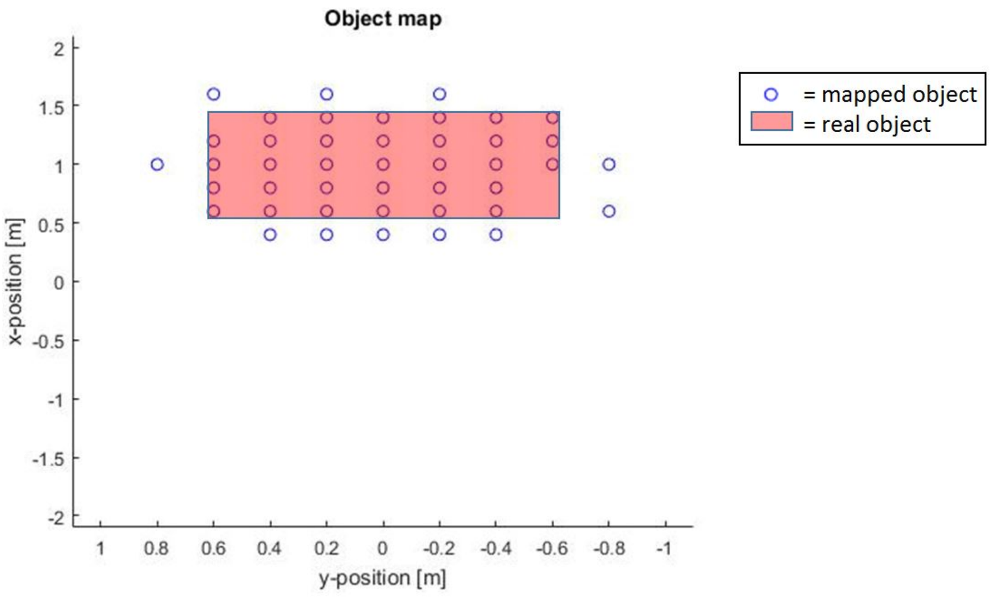


Figure 9.5: Map of the obstacles in static experiment

Although the mapped object matches the real situation, some deviations are noticeable. Some spots where the object is not present, are marked. This has, next to possible localization errors, two main reasons. The first reason is that inherent to the used algorithm, the measuring point is rounded up to 20 cm (D_{mp}). This means that deviations up to 10 cm are possible. The second major reason is the inaccuracy of the ultrasonic sensor due to the conical propagation of the ultrasonic wave. The fact that some spots are not marked while the object is present, is again inherent to the used algorithm. For each desired measuring point, a deviation up to 20 cm is allowed to go to the next measuring point. Figure 9.5 shows that (x,y)-coordinates (0.6,-0.6) and (0.8,-0.6) are missing in the map. A zoomed in version (figure 9.6) of figure 9.2 shows that there is no “analog” measuring point that corresponds to point (0.8,-0.6). This explains why point (0.8,-0.6) is not included in the obstacle map. For point (0.6,-0.6), all “analog” measuring points are on spots where the obstacle is not present, what explains why this point is not included as well.

As explained in chapter 6, the obstacle map should be extended by $2 \cdot D_{mp}$ to be sure that the whole obstacle is covered. To take into account the ignorance of the area outside of the scanning area, the obstacle map also needs to be expanded with a border of $2 \cdot D_{mp}$.

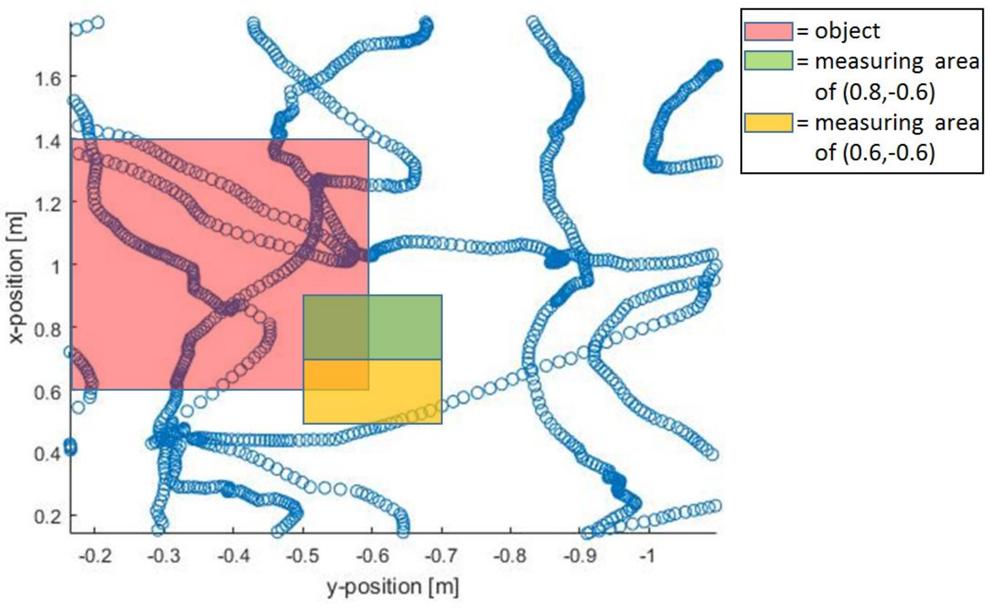


Figure 9.6: Trajectory of the drone in experiment with static obstacles (zoomed)

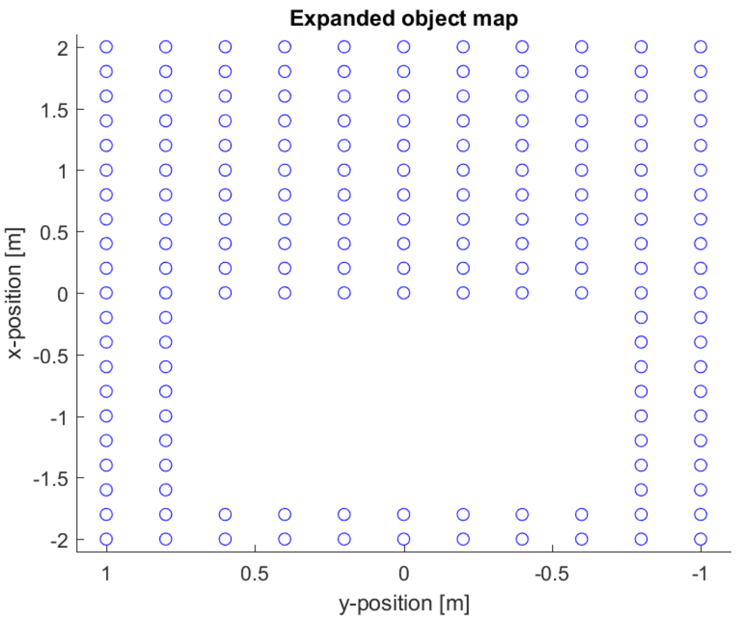


Figure 9.7: Expanded map of the obstacles in static experiment

Landing map

From the expanded map of the obstacles, possible landing spots can be derived by taking into account the radius of the drone. In figure 9.8, the landing map can be found where the blue circles indicate the spots where it is not possible to land.

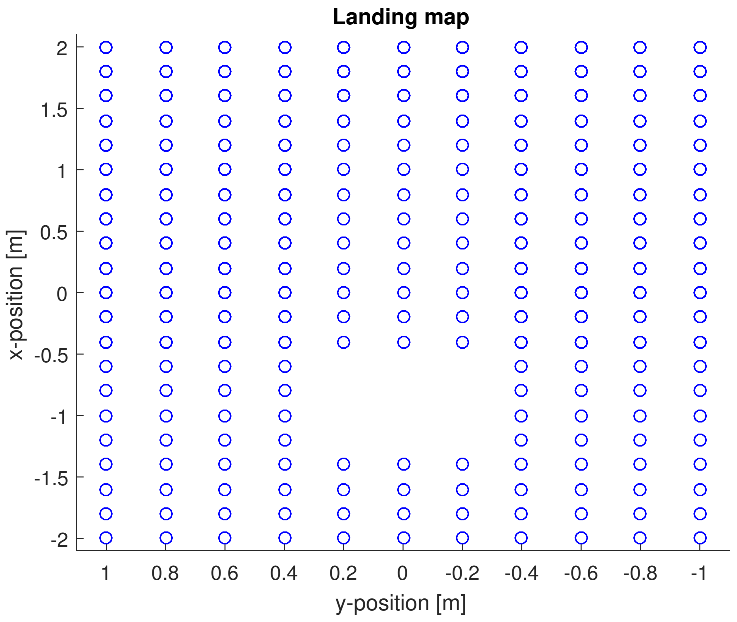


Figure 9.8: Map of possible landing spots in static experiment

Based on the target location, following optimal landing spot is obtained:

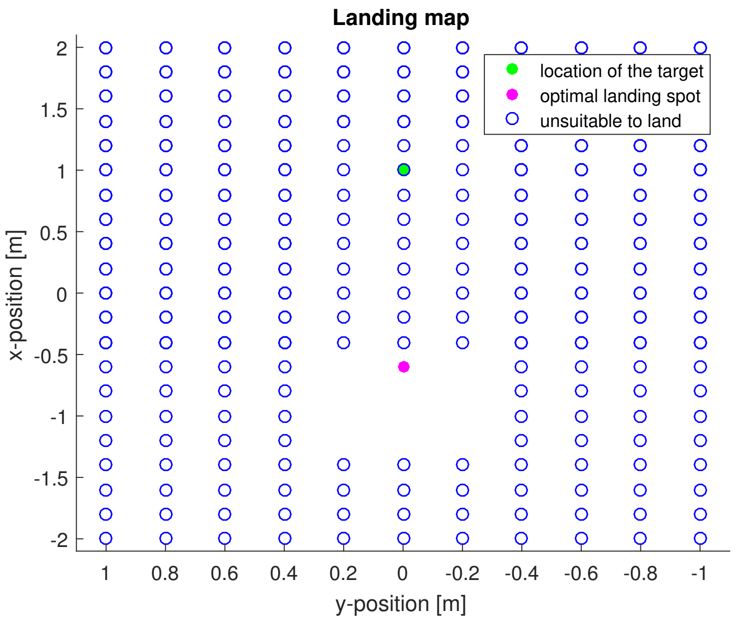


Figure 9.9: Optimal landing point in static experiment

The landing spot is about 1.2 meters away from the obstacle. This is due to the fact that the mapped obstacle is bigger than the real obstacle (0.2 m), the safety margin for worst-case scenarios ($2 \cdot 0.2\ m = 0.4\ m$), the finite dimensions of the drone (0.4 m) and the safety margin to buffer for limit cases (0.2 m).

Conclusion

The experiment shows the concept of autonomous landing based on distance meters. The intended accuracy has been achieved. Figure 9.3 proves that, even with the noisy ultrasonic sensor, detection of smaller obstacles is possible. However, experiments on this have not been performed since they would have little added value to the real-world application.

9.2 Moving obstacles

An initial approach to provide a proof of concept is through comparing the time a bot needs to go to a certain location (by using the velocity estimation of the bot) to the time the UAV needs to go to a certain location. When the time the UAV needs to land at that location is smaller than the time that the bot needs, the UAV can land on that location. Although this method really proves the concept of velocity estimations and similar experiments can be useful for the path planning, this experiments cannot be extended to landing in the real-world application. As one of the main concerns of an autonomous drone is safety, it is not recommended to assume constant velocity for real-world obstacles. The moving obstacles that an autonomous drone can face are very dynamical and can have fast changing speeds. That is the reason that this experiment rather focuses on the value of the velocity itself than on possible positions of the moving obstacle in the future. In this way it is possible to differentiate between cars moving on the highway at a speed of 120 km/h or cars that are in a traffic jam on the highway. As the drone in the real-world application will be equipped with sirens and signaling lights, one can assume that a car that is approaching the drone with a speed of 20 km/h will stop for the drone, while it is certainly not the case when a car is approaching with a speed of 120 km/h. It can be compared to the behavior of people who want to cross a zebra crossing. When a car is approaching with a velocity of 20 km/h, one can assume that the car will stop and people already start crossing the street. When a car is approaching with a velocity of 90 km/h, people are not going to cross the street for safety.

Indoor limitations

The idea behind the detection of moving obstacles is to interpret complex situations and to take appropriate decisions based on this additional information. Cars that stop or do not stop for an UAV landing, people making space for a drone landing and people approaching the drone to guide the landing are a few examples. The ideal proof of this concept would be to simulate these situations and perform an autonomous landing by interpreting these complex situations using speed measurements and obstacle recognition. Due to the limited achievable height of the UAV in indoor environments, only a small field of view of the bottom camera ($\sim 2m^2$) is achievable. An option to expand the field of view would be to add a fisheye lens to the bottom camera. However, this would not succeed. Apart from the fact that the quality of the bottom camera is too poor to apply this technique, the internal velocity measurements that are used in the internal controller of the drone would be strongly disturbed as those measurements are obtained by optical flow. As capturing of the above described complex situations is not possible with the current setup, the experiments focus on the estimation of the velocity itself and not on the interpretation of the situation.

Setup

The setup consists of a Lego Mindstorm that is driving with a constant speed of 0.2 m/s and a hovering drone that estimates the velocity of the bot using its bottom camera. In this experiment, one moving obstacle is considered.

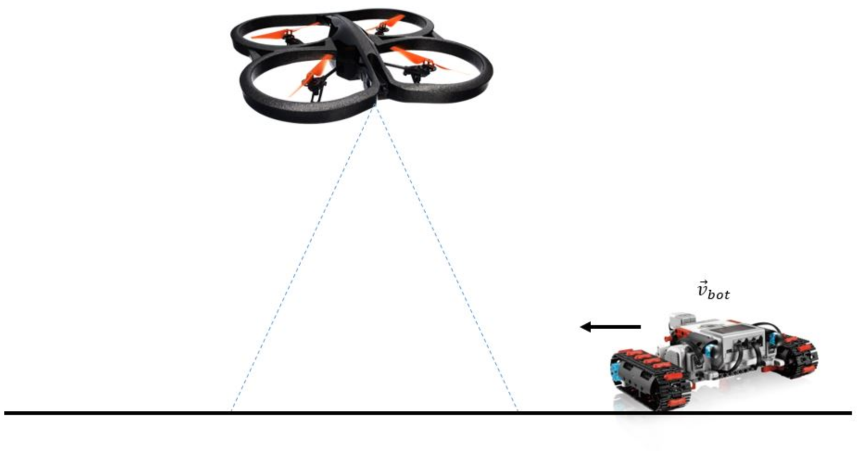


Figure 9.10: Setup for experiment on moving obstacles

Position estimation of the bot

In order to have a velocity estimation, a position estimation is required. In figure 9.11 and 9.12 the position of the drone, the position of the bot with respect to the drone and the (absolute) position of the bot is given. When the estimations are equal to zero, the robot is not in sight of the drone.

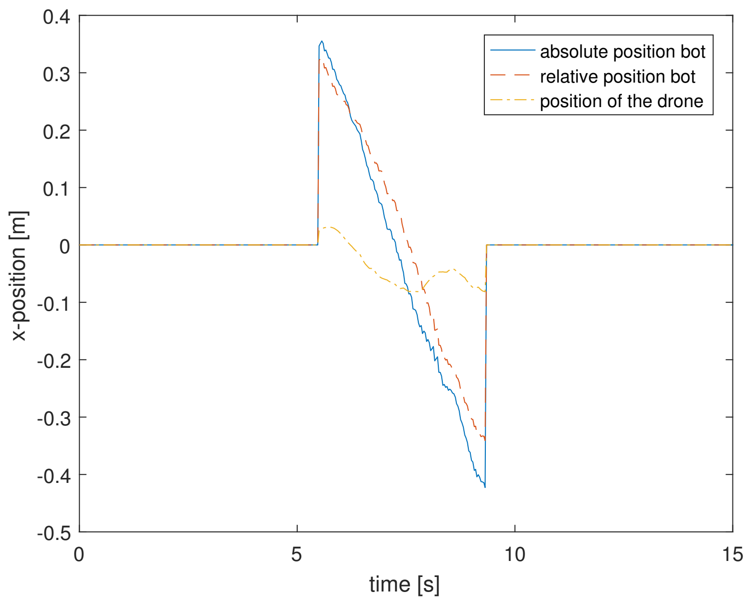


Figure 9.11: Position estimation of the robot in x-direction

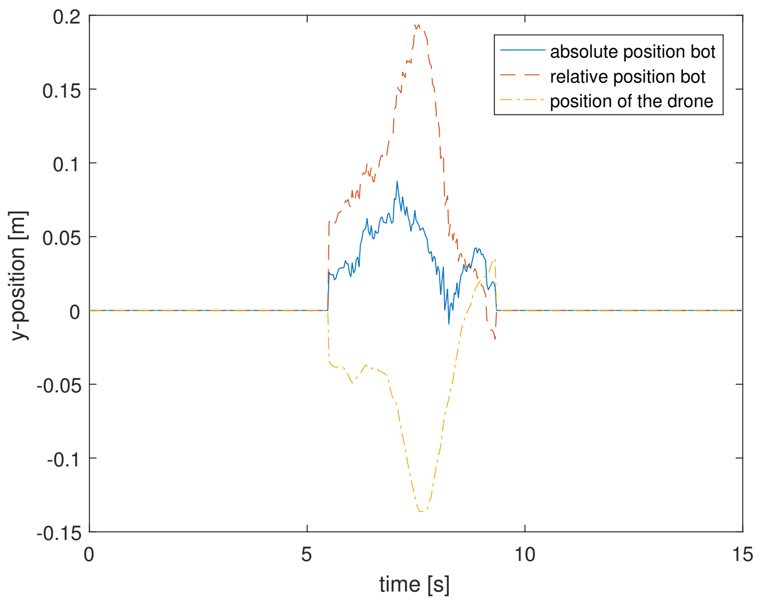


Figure 9.12: Position estimation of the robot in y-direction

The x-position is linearly dependent of time what corresponds to a constant velocity. The y-position fluctuates between zero and 0.1 m, which indicates that the real deviation of the y-position is low. The fluctuations are mainly caused by the imperfect localization of the drone, the delay between localization of the drone and the position estimation of the bot by camera and the fact that samples are taken in discrete intervals. These shortcomings mainly affect the estimation when the drone is very dynamic.

Velocity estimation

To have an idea of the measurement time that is required to have a proper velocity estimation, a cumulative moving average filter is applied. The result can be found in figure 9.13.

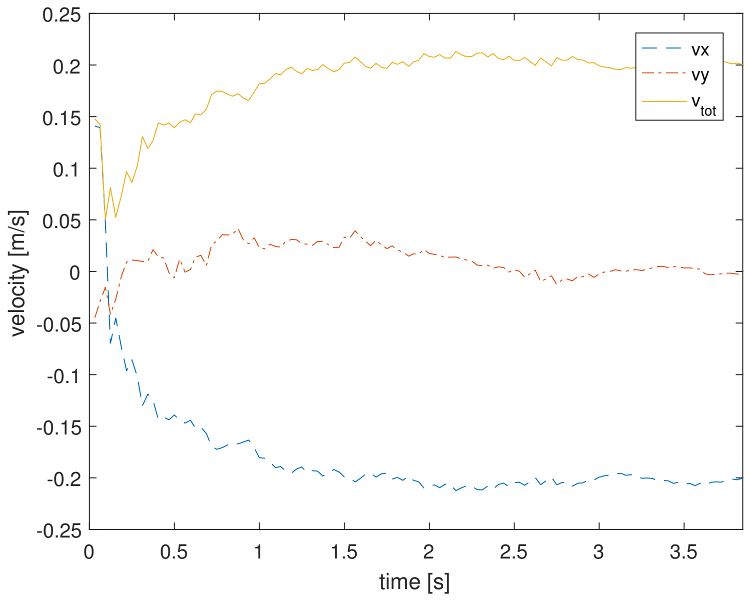


Figure 9.13: Cumulative moving average filter that is applied on the velocity measurements of the moving obstacle

This filter shows that the measurement is accurate (error is smaller than 0.015 m/s) from 1.5 seconds. This value is used to apply a moving average filter to the measurements. The implementation of this filter results in speed values between 0.185 and 0.240 m/s.

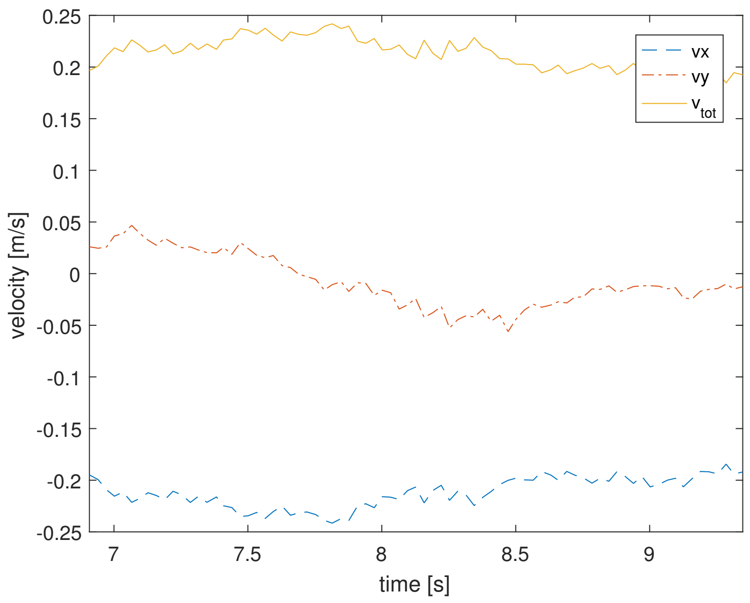


Figure 9.14: Moving average filter of 1.5 s that is applied on the velocity measurements of the moving obstacle

Conclusion

The experiment shows that velocity estimations based on vision is feasible from a drone. Even without camera calibration or taking delays into account, accurate velocity estimations of slowly moving obstacles are possible. This speed information can help autonomous UAVs to interpret complex situations.

Chapter 10

Conclusion

10.1 Achievements

This thesis addresses some possible problems of an autonomous drone. A landing algorithm that considers static obstacles and an algorithm for velocity estimations of dynamic obstacles, are applied to the drone. The basic ideas were experimentally proven, what demonstrates their feasibility for the real-world application. Next to the experiments, several other requirements of an autonomous drone are discussed.
The experiments are suitable for a first acquaintance with the requirements of an autonomous drone and they prove the concept of autonomous landing. Next to the conceptual elaboration, several options are suggested for the real-world application. These suggestions can provide ideas for researchers that are working towards the development of an autonomous drone or that are working on similar technologies.

10.2 Future work

The road to a fully autonomous drone is still very long. Since much research will be required, a few recommendations for future work are listed below.

Localization

The GPS offers a robust localization method, but it has to be investigated if this method is accurate enough. Certainly at landing, a high accuracy is required. A hybrid positioning system or SLAM might offer a solution to this problem. In case of the application of cardiac arrests, the bluetooth and WiFi signal of the cellular phone of the caller can help to find an accurate estimation of the target location.

Controller

Although the used drone was not very stable in the experiments, it is not the main control problem to let a drone just hover at a fixed position in indoor environments. Even the unmodified version of the used drone can hover stable at a fixed position. Research should be conducted on trajectory tracking control and fast rejection of disturbances resulting from stormy weather.

Obstacle detection

The accuracy, computation time and other features of different obstacle detection methods, should be investigated. These methods can then be evaluated in real outdoor environments. Research is also required for the velocity estimations. Velocity can be estimated as in the dissertation by differentiating the position, but it can be estimated by the Doppler effect as well.

Information about the obstacle

The information about the obstacle should not be limited to the position and velocity of an obstacle. Additional information can enhance the safety drastically. An autonomous drone should be able to differentiate between land and sea, but also between obstacles that are controlled by humans such as cars or the humans themselves and animals which have a more unpredictable behavior. Inter alia, multispectral sensors[1] and vision-based obstacle recognition are options to consider.

On the way to the target location

On the way from point A to B, another kind of environment is encountered than the environment encountered during the landing. While an UAV can hover some time above the target location during landing until an optimal landing spot is found, an UAV needs to take real-time decisions on its way in order to avoid dynamic obstacles. Thorough tests of path planning methods in presence of dynamic obstacles in less constrained environments should be performed.

Human guidance

The possibilities of humans that guide the drone to land should be examined. In this way the drone can land faster and in a safer way. As time is a very important factor for the cardiac arrests application, it can be useful that drones can eventually be guided by humans through windows of skyscrapers in urban environments. This application would require a very precise localization.

Database

Although obstacle detection is an indispensable feature of an autonomous drone, a lot of unsuitable landing spots already can be mapped in databases. In this way, a lot of unsafe environments can already be omitted beforehand. Methods to map these unsafe environments or methods to extract this information from existing maps of the world, should be further explored.

Future testing

For more extensive proof of concepts that include landing in presence of dynamic obstacles, more realistic ground situations should be simulated. Among others, human behavior should be implemented in the bots. Future research to create proper testing environments is needed. One of the options to consider is augmented reality[29].

Bibliography

[1] *Amazon develops active collision avoidance systems for its delivery drones.* URL: `https://patentyogi.com/latest-patents/amazon/amazon-develops-active-collision-avoidance-systems-delivery-drones/` (visited on 05/04/2017).

[2] *Amazon Prime Air.* URL: `https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011` (visited on 05/14/2017).

[3] Roman Barták, Andrej Hrasko, and David Obdrzálek. “On Autonomous Landing of AR. Drone: Hands-On Experience.” In: *FLAIRS Conference.* 2014.

[4] Jeffrey Byrne, Martin Cosgrove, and Raman Mehra. “Stereo based obstacle detection for an unmanned air vehicle”. In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.* IEEE. 2006, pp. 2830–2835.

[5] Andrew J Davison et al. “MonoSLAM: Real-time single camera SLAM”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007).

[6] Bram Diericx. “Evaluation of path planning algorithms for a quadcopter during indoor flight”. Master thesis. Ghent University, 2014-2015.

[7] Travis Dierks and Sarangapani Jagannathan. “Output feedback control of a quadrotor UAV using neural networks”. In: *IEEE transactions on neural networks* 21.1 (2010), pp. 50–66.

[8] Dave Ferguson and Anthony Stentz. “Field D*: An Interpolation-based Path Planner and Replanner”. In: *Springer Tracts in Advanced Robotics* 28 (2007).

[9] Shuzhi Sam Ge and Yan Juan Cui. “New potential functions for mobile robot path planning”. In: *IEEE Transactions on robotics and automation* 16.5 (2000), pp. 615–620.

[10] Jason Geng. “Structured-light 3D surface imaging: a tutorial”. In: *Advances in Optics and Photonics* 3.2 (2011), pp. 128–160.

[11] *Google Tango.* URL: `https://developers.google.com/tango/` (visited on 04/29/2017).

[12] John Paulin Hansen et al. “The use of gaze to control drones”. In: *Proceedings of the Symposium on Eye Tracking Research and Applications.* ACM. 2014, pp. 27–34.

[13] Felix Heide et al. “Doppler time-of-flight imaging”. In: *ACM Transactions on Graphics (ToG)* 34.4 (2015), p. 36.

[14] Andres Hernandez et al. “Identification and path following control of an AR. Drone quadrotor”. In: *System Theory, Control and Computing (ICSTCC), 2013 17th International Conference*. IEEE. 2013, pp. 583–588.

[15] Andres Hernandez et al. “Towards the development of a smart flying sensor: Illustration in the field of precision agriculture”. In: *Sensors* 15.7 (2015), pp. 16688–16709.

[16] Stefan Hrabar et al. “Combined optic-flow and stereo-based navigation of urban canyons for a UAV”. In: *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE. 2005, pp. 3309–3316.

[17] Yong K Hwang and Narendra Ahuja. “A potential field approach to path planning”. In: *IEEE Transactions on Robotics and Automation* 8.1 (1992), pp. 23–32.

[18] *Jeremy Clarkson reveals new prototype drones for Amazon Prime Air deliveries.* URL: `http://www.dailymail.co.uk/news/article-3338953/Jeremy-Clarkson-reveals-new-prototype-drones-Amazon-Prime-Air-deliveries.html` (visited on 05/14/2017).

[19] Andrew Johnson, James Montgomery, and Larry Matthies. “Vision guided landing of an autonomous helicopter in hazardous terrain”. In: *Robotics and automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE international conference on*. IEEE. 2005, pp. 3966–3971.

[20] *Lego shop.* URL: `https://shop.lego.com/` (visited on 03/20/2017).

[21] Carol Martínez et al. “On-board and ground visual pose estimation techniques for UAV control”. In: *Journal of Intelligent & Robotic Systems* 61.1 (2011), pp. 301–320.

[22] Geoff Gordon Anthony Stentz Maxim Likhachev Dave Ferguson and Sebastian Thrun. “Anytime Dynamic A*: An Anytime, Replanning Algorithm”. In: *Autonomous Robots* (2005).

[23] Tim G McGee, Raja Sengupta, and Karl Hedrick. “Obstacle detection for small autonomous aircraft using sky segmentation”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 4679–4684.

[24] Afshin Mohammadi, Meysam Rahimi, and Amir Abolfazl Suratgar. “A new path planning and obstacle avoidance algorithm in dynamic environment”. In: *Electrical Engineering (ICEE), 2014 22nd Iranian Conference on*. IEEE. 2014, pp. 1301–1306.

[25] Alec Momont. “Drones for good”. In: (2014).

[26] Harold F. Murcia. “A quadrotor as remote sensor for on-line prole measurement during harvesting process”. Master thesis. Ghent University, 2013-2014.

[27] N. J. Nilsson. “Principles of Artificial Intelligence”. In: *Autonomous Robots* (1980).

[28] Jerry P Nolan et al. “European resuscitation council guidelines for resuscitation 2010 section 1. Executive summary”. In: *Resuscitation* 81.10 (2010), pp. 1219–1276.

[29] Shayegan Omidshafiei et al. “Measurable Augmented Reality for Prototyping Cyberphysical Systems: A Robotics Platform to Aid the Hardware Prototyping and Performance Testing of Algorithms”. In: *IEEE Control Systems* 36.6 (2016), pp. 65–87.

[30] *Parrot AR.Drone 2.0 elite edition.* URL: `https://www.parrot.com/us/drones/parrot-ardrone-20-elite-edition#parrot-ardrone-20-elite-edition` (visited on 03/20/2017).

[31] *Pozyx accurate positioning.* URL: `https://www.pozyx.io/` (visited on 03/20/2017).

[32] Aaron Pulver, Ran Wei, and Clay Mann. “Locating AED enabled medical drones to enhance cardiac arrest response times”. In: *Prehospital Emergency Care* 20.3 (2016), pp. 378–389.

[33] Xue Qian et al. “Dynamic obstacle avoidance path planning of UAVs”. In: *Control Conference (CCC), 2015 34th Chinese.* IEEE. 2015, pp. 8860–8865.

[34] Matteo Ridolfi et al. “WiFi ad-hoc mesh network and MAC protocol solution for UWB indoor localization systems”. In: *Communications and Vehicular Technologies (SCVT), 2016 Symposium on.* IEEE. 2016, pp. 1–6.

[35] Giuseppe Ristagno, Tommaso Pellis, and Yongqin Li. “Cardiac arrest and cardiopulmonary resuscitation: starting from basic science and bioengineering research to improve resuscitation outcome”. In: *BioMed research international* 2014 (2014).

[36] Roberto Sabatini, Alessandro Gardi, and M Richardson. “LIDAR obstacle warning and avoidance system for unmanned aircraft”. In: *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering* 8.4 (2014), pp. 718–729.

[37] Satoshi Suzuki et al. “Topological structural analysis of digitized binary images by border following”. In: *Computer vision, graphics, and image processing* 30.1 (1985), pp. 32–46.

[38] Maxim Likhachev Sven Koening. “D* Lite”. In: *AAAI-02 Proceedings* (2002).

[39] Wikimedia Commons. *De HSV-kleurruimte voorgesteld als kegel.* 2005. URL: `https://commons.wikimedia.org/wiki/File:HSV_cone.jpg` (visited on 02/04/2017).

[40] Wikimedia Commons. *Relatie RGB-waarde met de daarbij passende kleurweergave.* 2004. URL: `https://commons.wikimedia.org/wiki/File:RGB_farbwuerfel.jpg` (visited on 02/04/2017).

[41] Peng Yao, Honglun Wang, and Zikang Su. “Real-time path planning of unmanned aerial vehicle for target tracking and obstacle avoidance in complex dynamic environment”. In: *Aerospace Science and Technology* 47 (2015), pp. 269–279.

[42] Chang-Sun Yoo Chang-Sun Yoo and Iee-Ki Ahn Iee-Ki Ahn. “Low cost GPS/INS sensor fusion system for UAV navigation”. In: *Digital Avionics Systems Conference, 2003. DASC’03. The 22nd.* Vol. 2. IEEE. 2003, 8–A.

Appendix A

Software

Programs for UAV

The programs for the drone are written in C++ using Microsoft Visual Studio. At the takeoff of the drone, an initialization is performed. The value of the Pozyx should be displayed as it is important that the drone already gets the values from the Pozyx before initialization. It is also important to put the drone in the same direction as the x-axis of the Pozyx, as the yaw angle of the drone is also re-initialized for each flight. In this way the x-axis of the drone (after coordinate transformation) and the x-axis of the Pozyx will coincide. For the reading of the Pozyx, the remote option is used. In order to read the position, a Pozyx device needs to be plugged on an Arduino. This Arduino needs to be connected to the computer during flight to transfer the values from the Pozyx to the main program via USB.

Program for Lego Mindstorm robots

The program for the Lego Mindstorm robots is written in Java using Eclipse. This program needs to be uploaded to the EV3 brick.

Program for Pozyx

The program of the Pozyx is written in C++ using the Arduino Software (IDE). This software needs to be uploaded to the Arduino that is connected with the computer.