TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING
MECHATRONICS AND AUTONOMOUS SYSTEMS CENTRE

Design and Implementation of a Robotized Drone Positioning System

Robotiseeritud droon-positsioneerimissüsteemi disain ja rakendus

MASTER THESIS

Student: Georgios Tymbakianakis

Student code: 184671MAHM

Supervisor: Prof. Mart Tamre

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

".....” ..... 201…..

Author: .....
/signature /

Thesis is in accordance with terms and requirements

".....” ..... 201….

Supervisor: ….....
/signature/

Accepted for defence

".....”.....201… .

Chairman of theses defence commission: .....

/name and signature/

Non-exclusive Licence for Publication and Reproduction of

GraduationTthesis[1]

I, Georgios Tymbakianakis (name of the author) (date of birth: 28.02.1990)

hereby

1. grant Tallinn University of Technology (TalTech) a non-exclusive license for my thesis Design and Implementation of a Robotized Drone Positioning System (title of the graduation thesis)

supervised by Professor Mart Tamre (Supervisor's name)

1.1 reproduced for the purposes of preservation and electronic publication, incl. to be entered in the digital collection of TalTech library until expiry of the term of copyright;

1.2 published via the web of TalTech, incl. to be entered in the digital collection of TalTech library until expiry of the term of copyright.

1.3 I am aware that the author also retains the rights specified in clause 1 of this

license.

2. I confirm that granting the non-exclusive license does not infringe third persons'

intellectual property rights, the rights arising from the Personal Data Protection Act or

rights arising from other legislation.

[1] Non-exclusive Licence for Publication and Reproduction of Graduation Thesis is not valid during

the validity period of restriction on access, except the university`s right to reproduce the thesis

only for preservation purposes.

_____ (signature)

_____ (date)

MECHATRONICS AND AUTONOMOUS SYSTEMS CENTRE

THESIS TASK

Student: Georgios Tymbakianakis 184671MAHM (name, student code)

Study programme, main speciality: MAHM02/18 - Mechatronics (code and title)

Supervisor(s): Professor, Mart Tamre, 620 3202 (position, name, phone)

Consultant: ………………………………………..…….....…….......... (name, position)

……...………….....…......….……………………………………… (company, phone, e-mail)

Thesis topic:

(in English) Design and Implementation of a Robotized Drone Positioning System

(in Estonian) Robotiseeritud droon-positsioneerimissüsteemi disain ja rakendus

Thesis main objectives:

1.Create a system that detects a drone with the use of machine vision

2.With the use of the xArm 7, translate the position of the drone to robotic movement

3.Pick and place the drone in a pre-defined position

Thesis tasks and time schedule:

No	Task description	Deadline
1	Setup xArm and Python	20.3.2020
2	Test and Configure Camera	30.3.2020
3	Detection and Translation Algorithm	20.4.2020
4	Design grapping fixture	20.4.2020
5	Testing	01.5.2020
6	Result summarizing and Report Compiling	19.5.2020

Language: English

Deadline for submission of thesis: 25.05.2020

Student: Georgios Tymbakianakis “….”………..……......…..201….a

/signature/

Supervisor: Professor Mart Tamre “…..”………..……..…….201….a

/signature/

Head of study programme: Professor Mart Tamre “….”……………………..201….a

/signature/

Terms of thesis closed defence and/or thesis restricted access conditions to be formulated on the reverse side

Contents

PREFACE

The idea for this thesis was proposed by Professor Mart Tamre and the research and implementation was done in the Lab of the department of Electrical Power Engineering and Mechatronics of the Tallinn University of Technology. Further help and guidance were provided by Mr. Dhanushka Chamara Liyanage, PHD student in the department of Electrical Power Engineering and Mechatronics.

The author would like to express his gratitude to the Mechatronics Programme Director, Professor Mart Tamre, for providing all infrastructure and materials needed to conclude this thesis. The Author also wants to thank his colleague, Mr. Michael Casey Roberts, for his support throughout the conduction of this thesis.

Keywords: robotics, machine, vision, drone, detection

PREFACE IN ESTONIAN LANGUAGE

Antud teema idee pakkus välja professor Mart Tamre. Uuringud ja rakendus sai läbi viidud Tallinna Tehnikaülikooli elektroenergeetika ja mehhatroonika instituudi laboris. Täiendavat abi ja juhendamist pakkus elektrotehnika ja mehhatroonia instituudi doktorant Dhanushka Chamara Liyanage.

Autor soovib tänada mehhatroonika magistriprogrammi juhti professor Mart Tamret, kes toetas lõputöö läbiviimist vajalike materjalide ja taristuga. Autor soovib samuti tänada tema kolleegi Michael Casey Roberts'it, kes pakkus oma tuge kogu töö läbiviimise aja jooksul.

List of abbreviations

UAV	Unmanned aerial vehicle
VRP	Vehicle routing problems
DoF	Degrees of freedom
QR	Quick Response
RGB	Red green blue
SDK	Software development kit
USB	Universal serial bus
TXT	Text extension
STL	Stereolithography
GUI	Graphical user interface

1. INTRODUCTION

An unmanned aerial vehicle (UAV) is an aircraft without a human pilot on board and can also be referred to as a Drone. UAVs are a component of an unmanned aircraft system which include a UAV, a ground-based controller, and a system of communications between the two. The first recorded UAV[1] was the used for the Austrian incendiary balloon attack on Venice in 1849(Figure 1.1).

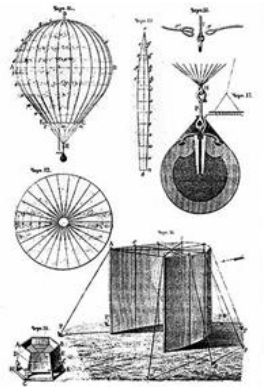

Figure 1.1 Austrian incendiary attack balloon on Venice [1]

The flight of UAVs can commence with various degrees of autonomy: either under remote control by a human operator or autonomously by onboard controllers. UAVs usage has attracted significant attention the past decade especially after the latest improvements in flight stability control [2] and composite materials allowing us to design and manufacture lightweight composite frames with high structural efficiency [3]. We have reached a point where Drones, with fully automated flying capabilities, weigh less that 5 kg (Figure 1.2).

Figure 1.2 DJI Mavic Air [4]

One area were UAVs have found growing grounds is the package delivery system where companies employ the use of drones to deliver payloads in an autonomous manner. The advances in this field, the past years, are extraordinary and we have managed to address task scheduling and path planning problems, for a team of cooperating UAVs performing autonomous deliveries in urban environments [5]. That has allowed companies, like Amazon, to implement small networks of automated drone delivery services where a UAV caries a payload and either deposits its payload close to the target area or on a specialized area where it is collected and processed by the company's personnel. The challenges associated with automated drone delivery systems arise from the need for maintenance and accuracy of landing as well as the development of solutions for vehicle routing problems (VRPs) specifically for drone delivery scenarios [6]. We have created solutions when it comes to wireless charging [7] and general maintenance however the precision of automated landing systems does not allow us to create a reliable automated drone positioning system in order to combine such solutions.

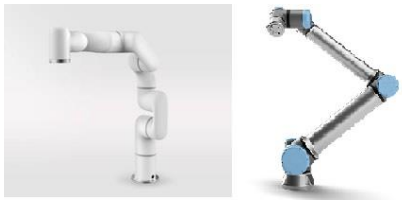

Figure 1.3 4999$ xArm 7 [8] and 56500$ UR10 robot [9]

The recent developments in robotics and machine vision tools have allowed companies to create hardware solutions such as cost-efficient robotic arms and stereoscopic cameras. For example, in Figure 1.3 we can see that a 4999$ 7DOF robotic arm can cost an order of magnitude less than a similar capability robot of the recent past. Granted, the use case of these 2 robots might be different, the prototyping and proof of concept capabilities are still there. It is in the hand of engineers now to combine all these different technologies and come up with cost effective solutions to arising problems.

2. LITERATURE OVERVIEW/ANALYSIS

2.1. Thesis Objective

The objective of this thesis is to create a system that will employ a robotic arm and a camera or set of cameras, in order to detect a drone with its payload in 3D space. Then after translating its position to mechanical movement, the robot arm will grasp the drone, with the use of its grip, by a custom-made bracket, that will also serve as a reference point. The bracket will incorporate a QR code that will provide a point of reference. Finally, the robotic arm will move the drone to a predefined location in a safe manner that will not compromise the structural integrity of the drone and the payload.

2.2. Research Objective

The objective of this research is to design and implement a precise post-landing positioning system that will physically manipulate a UAV, post-landing, in order to position it in such way that its payload can be deposited or that wireless charging can commence. This will be achieved with the use of a robotic arm and manipulator that will mechanically grab the drone and position it onto the desired location. The detection of the drone will be done optically using a depth sensing camera That will detect a QR code (Figure 2.1), located on the drone.

A QR code is an abbreviation for Quick Response code and it is a type of matrix barcode first designed in 1994 for the automotive industry. Its advantages include fast readability and greater storage capacity compared to standard UPC barcodes. Applications include product tracking, item identification, time tracking, document management, and general marketing [10].

The QR code could will provide information about the drone and could allow for possible expansion of the system. The system must be universal, meaning that the type of drone used should not affect the performance of the system. The system should also be cost effective and weatherproof. Therefore, the result of this thesis will be a HW and SW system that will be able to detect an UAV in 3D space, control a robotic arm in order to mechanically grasp the drone and manipulate it in order to place it in a precise location.

Figure 2.1 Example of a QR code

An important objective of this thesis is to find a solution to the perspective distortion issue that every pinhole camera has. Our X and Y coordinates will be detected by an RGB camera and therefore it will be subject to perspective distortion (Figure 2.2). We will either research ways to compensate for it or research and implement a solution to work around it using the robotic arm itself and the depth sensing camera in order to account for it mechanically.

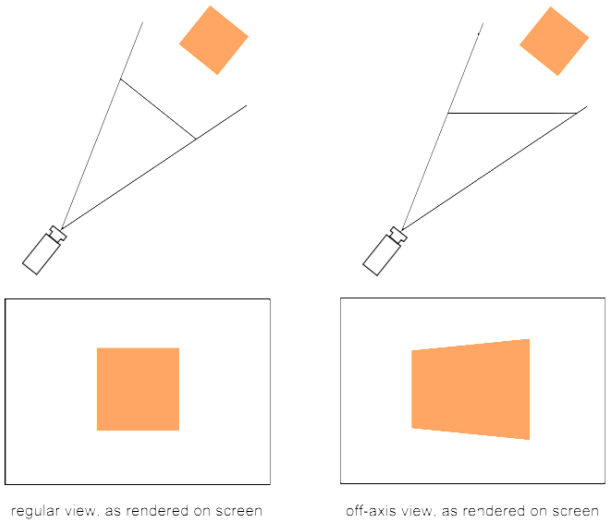

Figure 2.2 Perspective Distortion

Our research will also focus on how environmental factors affect the performance of such a system. It is well known and documented that different lighting conditions affect the

performance of the camera [11]. Lighting can affect due to reflectivity but also, depending on the spectrum of the light source (Figure 2.3), could also introduce artifacts in the RGB stream of a pinhole camera. This is the reason why we will end up testing our system under various lighting conditions both statically and dynamically.

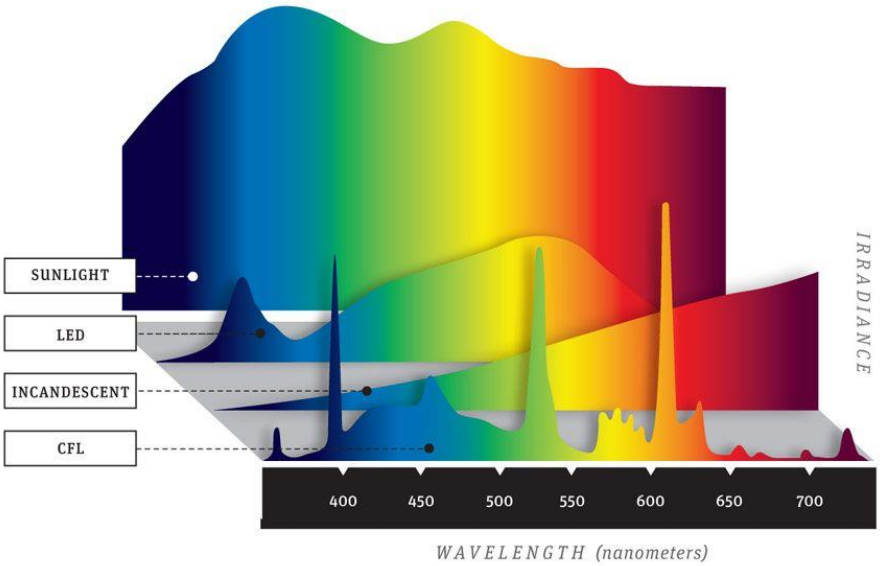

Figure 2.3 Comparison of Spectrum from different Light Sources [12]

As we see in the above figure, different illumination sources produce a different wavelength pattern. In the case of CFL, for example, we see peaks at various points in the spectrum. This could, in fact, alter the result of an RGB camera if the environment is not controlled. Various reflections could also affect the result of the image acquisition. Under strong illumination, glare could render a QR code unreadable. White balance of the camera's sensor could also increase the possibility of image clipping. In digital image acquisition, clipping is a result of capturing an image where the values in a certain area falls outside the maximum intensity which can be represented. As a result, information in the acquired image is lost. In Figure 2.4 we see an example of clipping where information is lost.

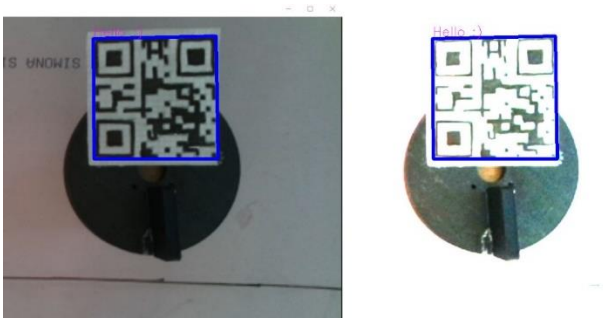

Figure 2.4 Example of Clipping

2.3. Existing Solutions

For years, companies such as Amazon and Google have been hard at work developing a safe and practical way of utilizing the potential of unmanned aerial vehicles to improve upon their current network of delivery services. Transportation of relatively large packages has become feasible due to the advances in drone-based robotics. The biggest problem left unanswered is how to achieve a safe release of the payload once the drone has arrived at the drop off location. Crude solutions have been achieved such as a basket that can hold onto the package until the drone is rapidly flipped upside down in an aerial maneuver where the centrifugal force ejects the payload from the basket [13].

Figure 2.5 Amazon PrimeAir Drone [14]

The need for accurate payload delivery stems from the problems that arise when we are trying to create automated post-landing payload handling. In other words, after a drone has landed, the payload should be retrieved in a safe manner without compromising or damaging the payload. Such control over the payload retrieval can allow us to further process they payload without human intervention. For example, a drone can deliver a package and the package can be retrieved and loaded in a cargo container [15], since the deposit of they payload will be accurate and there will be no need for human intervention in order to load or transfer the payload.

There are multiple ways to ensure an accurate landing position including the use of GPS systems and on-board sensors [16] but there is a need for off-board and universal precise landing systems that could support a multitude of different drones and be a "plug and play" solution.

The advances in Machine Vision and Machine Learning technology is a vital tool for creating such robust payload delivery platforms. Since UAVs undoubtedly pose several threats to airspace safety that may endanger people and property, several drone-detecting techniques have been developed the past years [17] however optical detection remains a cost-effective way to detect an object, in this case a drone. Recent developments in object detection algorithms also provide powerful tools with which we can tackle drone postlanding positioning problems [18].

2.4. Scope of Thesis

This thesis will explore 3 main research areas for which several problems should be solved in order to reach a novel solution. Several techniques will be considered but a single solution will be explored for each area.

First area is the Machine Vision and object detection in 3D space where with the use of a camera or combination of cameras we will be called to create an algorithm that will reliably detect our desired object in 3D space. We will have to explore solutions regarding the camera placement, relative to the landing pad. Several options will be investigated before we reach a conclusion of the camera placement.

The desired object will be a marker on the drone that will indicate an origin point for the system. This will require the exploration of a multitude of object detection techniques such as Region Shrinking [18] Stereoscopic imaging [19] [20]or Object Pruning [21]. In this context we will try to employ already existing techniques for optical detection [22] while adapting and possibly creating our own algorithm for object detection. Previous solutions

regarding picking and placing of objects with they use of a robotic arm have been found [23] [24] [25] [26] but in this case we need the tracking of the object to be dynamic and we need the system to adapt to changes such as a sudden change in the objects position due to weather conditions.

Second research area involves the coordinate translation. After we manage to detect our object in 3D space, we will use the cameras input stream to calculate the position of the joints in order to reach the desired object and grasp it. This is where the technique of choice for the object detection will determine how we will tackle this problem. If we use a stereoscopic imaging technique, one physical camera module is enough to detect the movement on the X and Y coordinate systems, but for movement the third dimension we need a second camera to detect the Z coordinate too. We will also investigate if forward or inverse kinematics is a viable option even though the magnitude of mathematical calculation involved is proven to be immense [27]. We will also explore the possibility of converting pixel coordinates into real world coordinates with the help of 2D transformation [28].

After we reach a decision on how the location will be obtained, we will have to employ they use of inverse kinematics in order to define the physical movement of the robotic arm. Inverse kinematics is a method that helps define the motion of a robot to reach a desired location. Another method of defining the motion of a robot is with the use of forward kinematics. In forward kinematics, the process of obtaining position and velocity of end effector is allowed given the known joint angles and angular velocities. In short, the difference between inverse and forward kinematics is that in inverse kinematics we define position and the process defines the angles of the joints of the robotic arm while in forward, we define the angles and the process defines the position. In this project we will research both processes.

The third research area is the Mechanical Engineering part where we would need to explore different solutions on order to mechanically tether the drone to the robotic arm. The engineering of a bracket that the manipulator will grasp will commence since it should not impede normal drone operation and it should be universal for all types of UAVs. We will experiment with different types of composites and conclude to a cost-effective way to engineer such bracket.

2.5. Tools Used

In this thesis we will use a suite of software as well as hardware in order to achieve our goal. Regarding hardware, the Mechatronics lab provided us with the xArm 7 robotic arm and intel's Realsence D435 camera. The xArm 7 robotic arm is a 7 DOF robotic arm manipulator designed and produced by uFactory.

Our programming language of choice is Python version 3.7. Within Python we will be using several libraries such as PYZBAR in order to decode QR codes. We will be programming in a windows environment using the PyCharm SDK. both our camera and robot arm come with their own API which we will use.

2.6. Outline

The thesis main body will be split into the several chapters exploring specific solutions. Each research area will have a separate chapter where a solution will be found.

In chapter 2 we will investigate the Machine Vision solution where we will decide our detection technique as well as our calibration process.

Chapter 3 will refer to the Coordinate Translation solution where we should manipulate and use the data acquired from chapter 2 in order to find a solution regarding the translation of said data into physical movement of the robotic manipulator.

In chapter 4 we will initiate a mechanical design for a bracket that will be used to tether the drone with the manipulator.

Chapter 5 will describe the combination of the solutions in order to achieve the desired effect. In chapter 5 we will also confirm that the combination of all the solution work as a unit.

In chapter 6 will define a testing methodology and perform several tests in order to reach conclusions regarding the reliability and robustness of the system.

In chapter 7 we will be describing future work and ideas that were not explored in this Thesis as well as limitations and suggestions for further improvement.

A final Chapter 8 will be used to summarize our conclusions.

Ultimately the thesis will contain 7 Chapters structured as seen below:

- Chapter 1: Introduction
    - Overview
    - Research Objective
    - Scope of thesis
    - Organisation of Thesis
- Chapter 2: Machine Vision
- Chapter 3: Coordinate Transformation
- Chapter 4: Mechanical Design
- Chapter 5: Final Implementation
- Chapter 6: Testing
- Chapter 7: Future Work
- Chapter 8: Summary and Conclusion

3. MACHINE VISION

Our first task was to write an algorithm in order to detect our object of interest, in our case the drone. The camera of choice for this project was the Intel RealSense D435 camera (Figure 3.1) that gives us a RGB camera sensor as well as a depth detecting sensor. With the use of the RGB sensor we can detect the location of the drone on the X Y axis while, with the depth sensor we can detect the distance of the drone from the camera. That will allow us, through manipulation of this data, to locate the drone's 3D coordinate relative to the camera. The Realsense camera also allows us to use only one camera module.

Figure 3.1 RealSense D435 Camera Used

Our proposed solution implements the detection of a QR code that will be placed on top of the drone thus giving us the ability to detect orientation as well. The QR code will be placed on top of a fixture that we will design in a future chapter. The fixture will be used to tether the drone with the robotic manipulator. The QR code may also contain data useful for identification or even for the implementation of security measures as future work. It is crucial for our design to be able to isolate a drone from the environment as well as detecting multiple drones in the same frame.

3.1. Design

In order to start designing the system, we must lay down the fundamental steps of the process. The first step, after the drone lands, is to be able to detect the QR code and decode its information. Using the PYZBAR library we can successfully detect the QR code

in the live video stream. After the QR code is detected, we can use its location in the image to determine its center point as well as its orientation. We can do that since PYZBAR can return the XY location of each corner of the QR code as well as the text data that the QR code carries.

We can also use a library to detect multiple QR codes in the image stream. This information will then be stored in order to be used in the coordinate transformation step described in chapter 3.

Important step in our design, is to make sure that we take perspective distortion into account and avoid errors in our calculations. In order to succeed in that we will Conduct our tests after we manually calibrate the position of the camera in order to avoid perspective distortion.

After we have managed to reliably detect the QR code in 3D space we will have to decide the placement of the camera. The camera should be placed in a way that it will not impede a drone's ability to land.

3.2. Implementation

3.2.1. Object detection

In order to implement our design, we initiated a new project in PyCharm and created a new Python File. We started by importing all or needed libraries. These include the Intel Realsense camera library as well as the PYZBAR library.

In this early stage and for process verification reasons, it is of great importance to calibrate the camera in such a way that it's perpendicular to the surface in order to avoid perspective distortion problems. The Realsense camera SDK and software suite provides us with a calibration tool that depicts the projected surface and its distance from the camera. below you can see the difference between before and after the calibration. The calibration was achieved by leveling the camera manually. In later chapters we will calibrate the camera through our software.

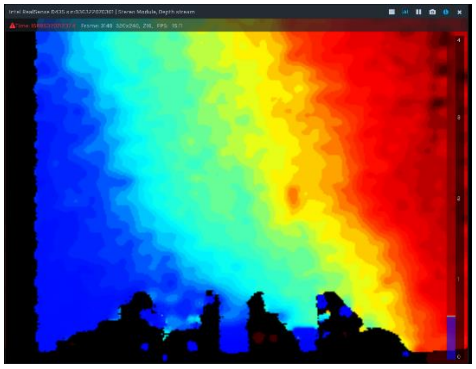

Figure 3.2 Depth Gradient Before Calibration

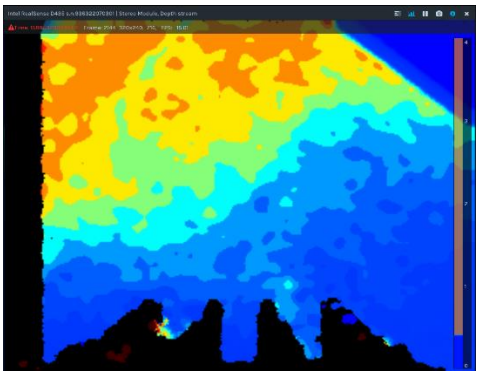

Figure 3.3 Depth Gradient After Calibration

After leveling the camera manually, we managed to go from a 0.2 cm slope two a 0.02 cm slope, which is an order of magnitude lower and therefore we can proceed with our calculations being confident that perspective distortion is not going to affect our calculations.

The next step was to initialize our camera feed using the openCV library. Since the Realsense camera is the only one actively connected to the computer, its identification code is 0. We stored each frame of the camera into a variable called IMG and with the use of the *decode()* function of the PYZBAR library we were able to store the QR code as an structure into a variable called *barcode*. This structure contains all information about the QR code. One of the items of this structure is *barcode.polygon* that contains an list with four 2 item lists where the XY coordinates are stored (3.1). We can pass that to a variable called *poly*. The coordinates correspond to each corner of the QR code in a counterclockwise direction, starting from the top left side of the code. We then saved each coordinate to a different variable in order to proceed to calculations.

$$poly = barcode.polygon = [[x0, y0], [x1, y1], [x2, y2], [x3, y3]] \quad (3.1)$$

With the use of the OpenCV library, we were able to display the feed from the camera while overlaying the position of the QR code in the image as well as displaying the text that the QR code contains. For prototyping purposes, we created a hand drawn QR code in order to demonstrate the robustness of the detection. In the future and during testing we will perform a robustness test and compare the hand drawn QR code with a printed one as well with a partially destroyed one.

Figure 3.4 Result of the Camera feed

Now that we have the X Y coordinates of each corner of the QR code we can employ basic math functions in order to find specific points relative to the QR code as well as finding the angle that the QR code is orientated. By adding the coordinates of two opposed corners of the QR codes and divided them by two, we can find the midpoint coordinates, *xPos, yPos* between these two points (3.2) and therefore the center of the QR code and with that information we can determine an offset where the gripper will grasp the drone from. Since in a later chapter we will be designing our own fixture we can use this to our advantage to achieve a standardized gripping procedure.

$$xPos = \frac{poly[0][0] + poly[3][0]}{2} \tag{3.2}$$

$$yPos = \frac{poly[0][1] + poly[3][1]}{2} \tag{3.3}$$

Since now we know all four points of the QR code we can easily calculate the angle of the QR point by choosing two points and using atan2 function(3.4). This returns the angle in radians. This value, as we will see later, can be used as radians or can be translated into degrees.

$$\theta = atan2(poly[1][1] - poly[2][1]\,, poly[1][0] - poly[2][0]) \tag{3.4}$$

Additionally, the PYZBAR library allows us to retrieve a bounding box with the area that the QR code covers(Figure 3.5 Bounding Box in Blue. That might be useful in following chapters in order to determine an area of exclusion.

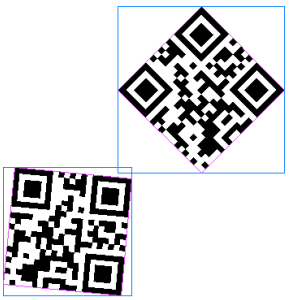

Figure 3.5 Bounding Box in Blue

In our next step, since we managed to record the camera feed, detect a QR code in it and extract its location in XY space, the next step is to detect its distance from the camera.

The real sense library allows us to save the stream from the cameras depth detection module and align it with the RGB feed. This aligning will allow us to refer to a specific point in the RGB feed that will correspond to the same physical point in the depth sensing stream. After we have received the coordinates we can use the *get_depth()* function, which returns the distance from the sensor to the specific point in meters. In order to test that, we placed the QR code on top of a box with known dimensions and took measurements. The camera was able to detect the difference with an accuracy of a millimeter (Figure 3.6).

Figure 3.6 RGB Stream aligned with Depth stream

Fortunately, the Realsense library makes it easy for us to detect accurate depth without making any calculations, since the *get_depth()* function returns a value in meters. That value can easily be translated into movement of the robotic arm since we have full control over the movement of the arm and we also know where it is located at any given time.

Finally, we had to come to a decision regarding the position of the camera in the system. At this point we have the camera mounted on an adjustable tripod in order to verify our detection. This will not be possible in the final set up since a camera mounted rigidly above the landing pad will prevent a drone's ability for vertical landing. We must keep in mind that the landing pad should not be obstructed by any physical object therefore the robot arm and the camera should be retracted prior to the detection. Another option was to rigidly mount the camera offset to the landing pad looking at an angle and then correct for distortion in the image. The best solution was to use of the bracket provided by the robotic arm manufacturer (Figure 3.7) in order to mount the camera on the end of the robotic arm and deal with the orthographic distortion in our next chapter.

Figure 3.7 xArm Camera Mount [29]

The bracket can be mounted in several different orientations allowing us a certain degree of freedom in our implementation. We decided to Mount it directly above and parallel to the grippers plane of interaction as seen in Figure 3.8. This will still introduce perspective distortion that we will have to account for. In later chapters we will see that there was no need for correction since we could implement a routine in the robotic arm movement in order to achieve a correct detection even though the camera is mounted offset of the landing pad. With this solution we will manage to not have any physical object over the landing pad and therefore a drone can land without any interference. This method also allows us to continuously monitor the video stream throughout the movement of the robotic arm. In the future this can be used to detect changes of the location of the QR code throughout the retrieval process.

Figure 3.8 xArm Camera Module Mounted on xArm [29]

We are now able to detect a QR code in 3D space, relative to the camera, and therefore in the next chapter where we will solve the problem of translating these coordinates into a pose for the robotic arm.

3.2.2. Landing pad calibration

Our Landing pad will consist of a flat piece of plastic where we will inscribe a parallelogram that will serve as our landing limits. This will define our region of interest. Everything out of this limit should be disregarded. The camera, once activated, records the RGB data of the scene but does not know what consists the limits of the landing pad. Therefore, there should be some connection between the cameras feed and the physical world. One way to achieve that is to detect the inscribed limits though the use of machine vision techniques. For example, we could apply a contrast filter and a threshold and perform rudimentary edge detection finding the inscribed line. That unfortunately proved unreliable since in a real-world application, the landing pad might not be defined or maybe there would be other artifacts in the image, such as glare or leaves. Another method is to perform a linear dependency calibration. This involves the placement of a detectable object in the corner of our region of interest and detecting its position. Then we can manually place our physical end effector to that position and record its position. We would then have a correlation between what the camera stream detects and its position in the physical world. This is achieved easily with the use of a dummy QR code. By placing the QR code on the limits of our defined landing pad We can detect those positions and their coordinates. That will allow us to set restrictions in our coordinate translation in order to detect whether a drone has landed within the landing pad.

An obvious problem that arises is that we do not have yet to define a default birds eye position for our robotic arm, therefore this calibration routine will be executed after the default birds eye pose of the robot is defined in a Chapter 3. The reason why this is a problem is because we are basing our limits on the *XY* data of the cameras RGB stream. Those reference points change based on the camera location relevant to the landing pad, due to perspective distortion (Figure 3.9). Since we have mounted the camera on the robot, that position should be standardized for our technique to work. We will assume that a standardized pose is set and that the cameras FOV covers the whole landing pad and that it is perpendicular to the surface of the landing pad (4.2).

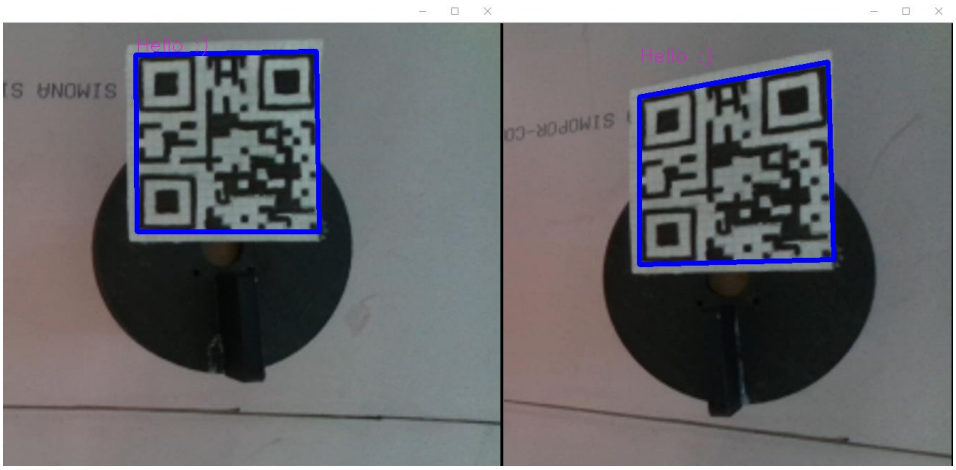

Figure 3.9 Perspective Distortion

Next step in the calibration process will be to correspond the corners of the landing pad with the position of the gripper of the robotic arm. This will allow us to achieve a linear relationship between the location of the QR code in the image and the physical location of the end effector. We will start by placing a calibration QR code on the top-right corner of the landing pad. Then, with the use of the PYZBAR library we can detect its location, as seen in Chapter 2.2.1. This will give us a struct object with the coordinates of each corner of the QR code. After we calculate the midpoint of the QR code we will have a reference point on the RGB stream of where the top and the right limit is.

After we obtain the position of the QR code in the RGB stream we can manually move the robotic arm to the center location of the QR code. This will allow us to associate the position on the RGB stream with the position of the robotic arm. With the xArm library we can obtain the position of the end effector and store the pose as joint angles.

Now that we have the location of the QR code, in the RGB feed, when the robot is in its Birds Eye position, and the location of the arm at that position we created a relationship between the location of the QR code and the location if the robotics arms end effector (Figure 3.10).

We repeat this process for the bottom right corner thus covering the whole landing pad. After we have all the calibration data, in the next chapter we will explain how with a simple linear interpolation function we can achieve an accurate detection of the drone's position.

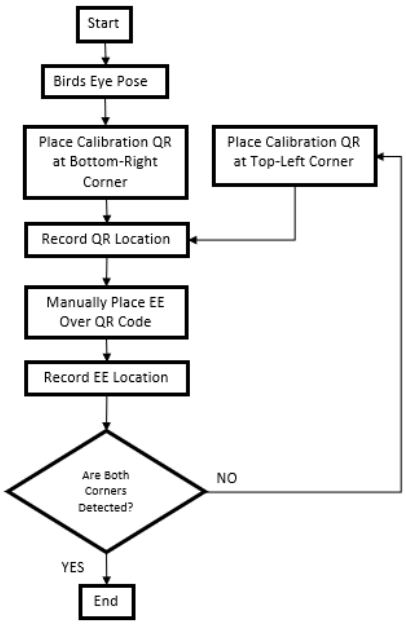

Figure 3.10 Calibration Process

It is important to understand that the camera and the robot have different coordinate systems. The cameras X coordinate defines the right-left position of the pixel and the Y coordinate defines the bottom-top position of the pixel. The robotic arm positions X coordinate defines the top-bottom position of the end effector while the Y coordinate defines the left-right position (Figure 3.11). The cameras X coordinate indicates the number of the horizontal pixel and the Y indicates the vertical pixel. In our specific application, we used the maximum resolution provided by the Realsense camera(3.5)(3.6) which is 1280 pixels by 720 pixels. This will allow us to utilize the camera to its fullest since the higher resolution will allow us to detect smaller QR codes.

$$Range_x = [0, 1280] \tag{3.5}$$

$$Range_y = [0, 720] \tag{3.6}$$

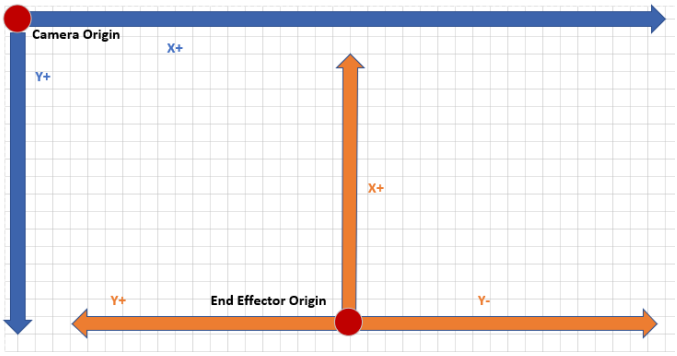

Figure 3.11 Combined coordinate system

Since we have recorded the edges of our landing pad and we have also recorded the position of the end effector at those positions we can achieve a linear dependence E as we desired. While the camera is at the Birds Eye view position and after we detect the position of the drone within the image, we can interpolate that position and converted into a XY position for the robotic arm. in or code we created our own data interpolation function that will take care of this conversion. The standard interpolation equation that we used is seen in (3.7).

$$Value_{x,y} = \left(\frac{QRx, y - LPt, l}{LPb, r - LPt, l}\right) * (RPy_{max}, x_{max} - RPy_{min}, x_{min}) + RPy_{min}, x_{min} \quad (3.7)$$

In our situation we are dealing with different coordinate systems therefore we must make sure that we are converting the correct data. Essentially we are trying to transform a value from one scale into a value on another scale and since we have aligned the camera perpendicular to the landing pad and perspective distortion is not a problem, we can do that with the linear interpolation equation (3.2). In order to control possible input range inversions or output range inversions we wrote a simple re-map function seen in A1 (lines 37-70).

4. COORDINATE TRANSFORMATION

Since in chapter two we managed to obtain a position of a QR code in 3D space, relative to the camera, our next task is to translate that into physical movement of the robotic arm. This will allow the robotic arm to move in 3D space and place itself and its gripper on top of the drone's fixture and grasp the drone with its grippers allowing it to transpose the drone in 3D space and place it in our desired location. That desire location could be a wireless charging pad or a storage unit. The importance of accurately translating the coordinates into movement is high, since we will consider the drone as a sensitive object and therefore will want to handle it with care. The accuracy of the placement after the pick-up of the drone is also important, especially for wireless charging. The reason is that wireless charging pads can charge only when the charging coil is within a range of some centimeters, therefore our placement should be accurate.

4.1. Design

The first step will be to define a default standby position for the robot arm. This standby position will have to not impede a drone's ability to land on the landing pad and should provide an angle that the camera will be able to overlook the entirety of the landing pad and even the surrounding area. Since now the camera can see our region of interest it can detect when a drone enters the landing pad region. After the drone lands, the robot will execute a pose, in order to place the camera over the landing pad and perpendicular to it, called the Birds Eye view position. It will then execute the detection algorithm described in A1 System Code and retrieve the needed XYZ coordinates. Then the decoding of the 3D coordinates we received from the camera will commence.

Our algorithm should be able to translate that data into a pose of the robot and execute that pose. This pose will land the grippers over our fixture and grip the drone. Then, the drone will be lifted and placed to the desired location. The desired location in our example will be hard coded but in future iterations this could depend on other parameters. For example, we could have different storage locations for different size drones. Another future solution could be that the drone will communicate with the system and the arm will place drones with low battery on the wireless charging pad. In Figure 4.1 Decision Tree we can see the decision tree of our system.

After the process is Complete, the drone should return to its standby position and enable its detection algorithm again in order to repeat all the steps above when the new drone flies in (Figure 4.1).

It is important in this stage of development to have a robust way of retrieving the *XYZ* coordinates and for that reason all physical components are rigidly mounted respectively to each other. This means that the relative position of the landing pad, the camera and the base of the robotic arm, is fixed

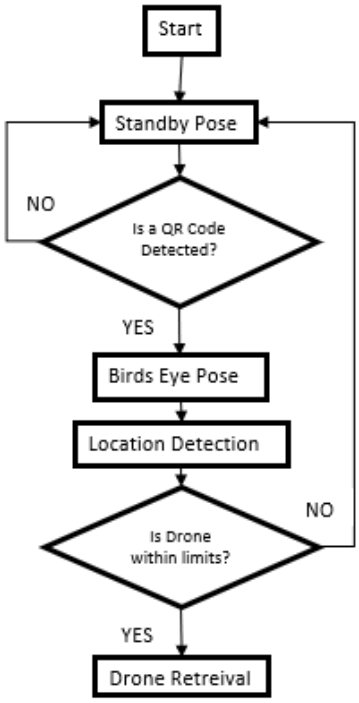

Figure 4.1 Decision Tree

As we can see the system is responsible to detect the presence of a drone and to detect if the drone is within the limits we have defined. We can also see that there are two predefined positions for the robotic arm, the Standby position and the Birds Eye position.

4.2. Implementation

4.2.1. Hardware

In order to begin implementing or solution, we had to set up the robotic arm with its gripper and define a landing pad in front of the robot. This also allowed us to finalize the physical location of the whole system and set up or computer workstation next to it. As mentioned in previous chapter 2, the Realsense camera was mounted at the end of the robotic arm and after the camera bracket the gripper was mounted. The gripper used is

the official xArm gripper by UFACTORY (Figure 4.2). This gripper is compatible arm library and we can control it through our script.

Figure 4.2 xArm Gripper -2019 [30]

The Realsense Camera is connected to the computer through a USB type C cable. The robotic arm is mounted on top of a sturdy metallic base and the landing pad is a sheet of plastic in front of it (Figure 4.3).

Figure 4.3 Workstation

4.2.2. Software

In our project folder in PyCharm we created a new Python file where we imported our needed libraries. In our case the xArm library was needed as well as several libraries that will help us with the coordinate transformation. In order to retrieve the data from the detection script, we inserted lines of code in the detection script in order to save those values in a TXT file on our hard drive. This was a temporary solution until we combine all our code under one Python file. The alternative was to create a pipeline between the two

files and through parallel programming obtain the data. Since in the end we ended up with one single file no such action was needed.

It is important in the software to initialize both the robot and the camera. The robot is connected with the use of an Ethernet cable through our network card. The xArm library allows us to send commands as: pose, angle, relative position and absolute position. we are also able to send commands to each of the seven joints of the robotic arm.

Next step is to define our standby position and the birds eye position and save them as an array (3.1) (3.2). As mentioned in chapter 2, the standby position is a position that will allow the robotic arm to be out of the way of the landing pad but will still enable the camera to have an overview of the landing pad. The Birds Eye position is a position that the robot will engage after it has detected a drone landing. This position will bring the camera perpendicular to the landing pad therefore allowing us to not have to correct for perspective distortion. The detection routine, mentioned in chapter 2, will commence after the robotic arm is in the Birds Eye position.

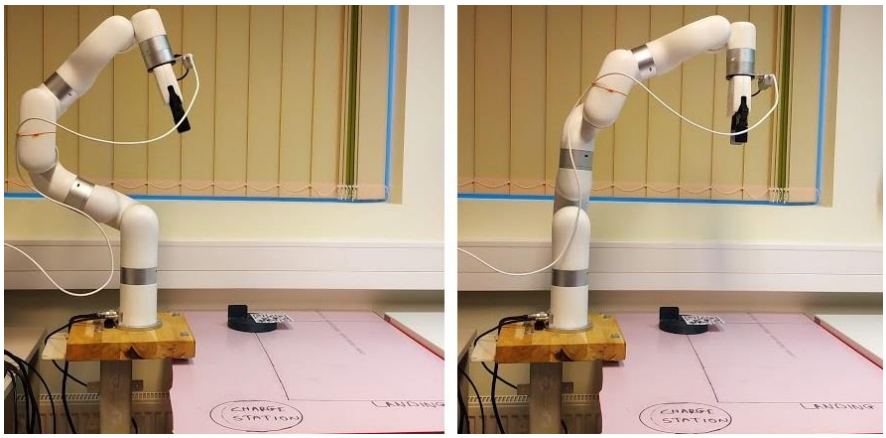

Figure 4.4 Robotic arm in Standby and Birds Eye pose

$$standBy = [0, -65, 0, 69, 0, 113.4, 0] \quad (4.1)$$

$$birdEye = [0, 3.3, 0, 116, 0, 112, 0] \quad (4.2)$$

Since the bird's eye position has been determined we can now proceed to the detection. Since the detection is done in birds eye mode, the maximum detection area is what the

camera can cover. The distance of the camera to the landing pad is limited by the ability of the camera to stay parallel to the landing pad. In our case the camera is located *h* = 70 cm over the landing pad and since the Realsense RGB sensor has an horizontal FOV of FOV_H = 91.2° and a vertical FOV of FOV_V = 65.5° [31], the maximum area of detection, that is the Area of View, is 143 cm(4.3) by 90 cm (4.4). In the calibration routine, described in 3.2.2, we mentioned how we achieved a linear dependency between the location of the QR code, in the RGB stream and the physical location of the end effector. Now, since we have all the required data, we can linearly interpolate the XY position of the RGB stream and the XY pose of the robotic arm. This is achieved by declaring our found limits (described in 2.2.2) as integer variables. There are four variables for the camera values and four for the robotic arm values. These four variables correspond to the topmost the bottommost the rightmost and the leftmost positions.

$$Hmax = 2\left(h \cdot \frac{sin\left(\frac{FOVh}{2}\right)}{sin\left(180 - (FOVh + 90)\right)}\right) \approx 143 \tag{4.3}$$

$$Vmax = 2\left(h \cdot \frac{sin\left(\frac{FOVv}{2}\right)}{sin\left(180 - (FOVv + 90)\right)}\right) \approx 90 \tag{4.4}$$

We must keep in mind that the above calculations are possible since the camera is parallel to the landing pad and the FOV is perpendicular to the landing pad.

As we mentioned in the previous chapter, since we have achieved linear dependency, we now must input our recorded XY position into the linear transformation function we created called *remap()*. We do that separately for X and separately for Y.

As an example, let's say that we detected in our calibration, that the limits of or landing pad in the camera's X axis (left-right) is from 379 to 1127. We also detected that the robotic end effector limits, Y axis (also left-right as seen in Figure 3.11) is 240 until -303. We are now able to retrieve the QR codes position, within the limits of the landing pad. This position is stored in two variables (*xPos, yPos*). When that value is retrieved we can input it into the *remap()* function in order to get a value back within the limit of the end effector movement (4.7).

$$camL = 379, \quad camR = 1127 \tag{4.5}$$

$$xArmL = 240, \quad camR = -303 \tag{4.6}$$

$$xLoc = remap(xPos, camL, camR, xArmL, xArmR) \tag{4.7}$$

Since we also have the angle of the QR code and therefore the orientation, we can insert that number as an angle for the joint that controls the end effector. The xArm library allows us to input that information both as radians and as angles. In order to be consistent, we will be using angles throughout this whole project. This also means that we have to convert the value that we found in equation (3.4) from radians to degrees(4.8). We can do that by using the math library in Python(4.9).

$$angle = \theta \frac{180°}{\pi} \tag{4.8}$$

$$angle = math.degrees(\theta) \tag{4.9}$$

We also showed in chapter 2 how we can obtain a depth reading and since this reading is in meters, we can give a relative movement command to the robotic arm. We can do that because we have defined the bird's eye view. The xArm library allows us to get the location of the end effector at any given moment and therefore we know the distance of the end effector from the landing pad. For example, if the QR code is detected in location *xPos, yPos* = [440, -370] we can retrieve the depth value by using the Realsence command *get_depth()*. That will provide us with a depth value in meters for the position we input.

Now we have all the information needed to input the pose to the robotic arm and let it grasp the drone. We can do this through the set servo angle command that the xArm library provides us. After the end effector reaches big grasping point of the fixture, we can set the gripper position, with the set gripper position command, and now we are ready to move the drone.

Before we position the drone to the predefined location, we lift the drone off the table and move it while keeping it level in order to avoid collisions with other drones on the landing pad. in order implementation we have set an area next to the landing pad that will serve as our destination. That will later allow us to calculate the accuracy of the positioning command.

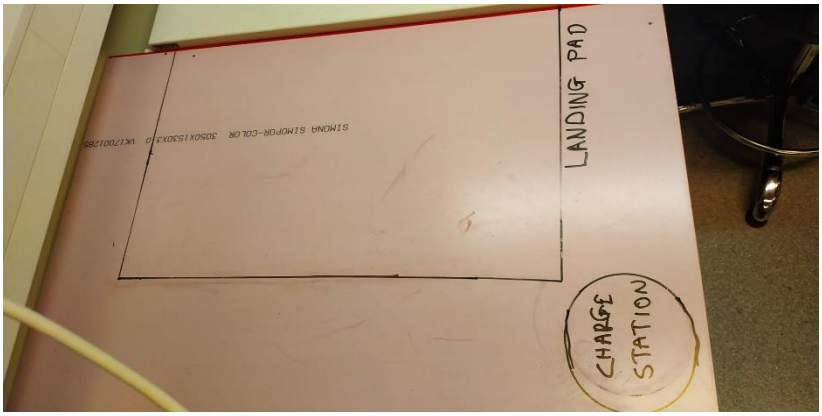

Figure 4.5 Landing Pad with Carging Station on the bottom right

In order to test if the system is working as intended, we temporarily fixed a QR code on a piece of 3D printed scrap with an upright post that will act as our gripping point. we will later design a gripping fixture.

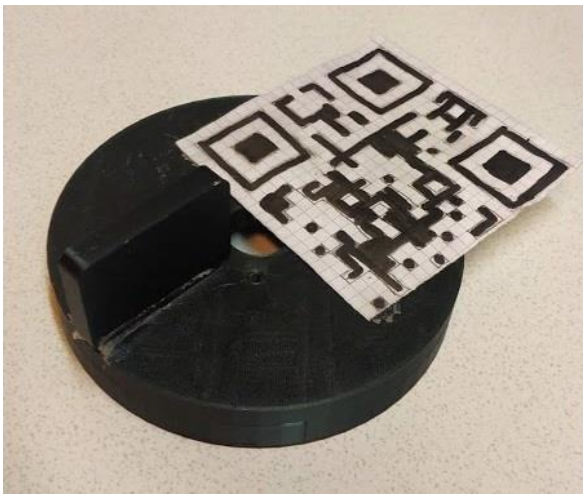

Figure 4.6 Temporary Fixture with Temporary QR code

Now that we have achieved the detection of the drone and the coordinate translation, pick up and positioning of the drone we can proceed to the next chapter where we will design the gripping fixture. After that we are ready to test the system.

5. MECHANICAL DESIGN

In this chapter we will find a solution for mechanically linking the end effector of the robotic arm with the drone. This fixture needs to be robust and needs to be able to withstand the load of a drone. We are limited by the carrying capacity of the robotic arm, which in our case is 3,5 kg. In addition, since our gripper is equipped with parallel grips, need to create an upright post and we will also have to test if the gripper is able to withstand that load. We will test the load capacity with the use of test weights that will be attached to our fixture.

5.1. Design

For the design gripping fixture, we will use Autodesk's Fusion 360 software with an educational license. This will allow us to design a fixture and export an STL file so that we can 3D print the fixture for further testing.

On top of the fixture, a QR code will be placed. A QR code is a matrix-type 2D code that has a 25 by 25 grid of elements in an image to be read. With that being said, the theoretical minimum size, given our area of view and resolution, is a QR_{min} = 2,79 cm square.

$$\text{QRmin} = \left(\frac{Hmax}{1280}\right) 24 \approx 2{,}792 \tag{5.1}$$

After trial and error, we have determined that the best size for a QR code is 10 cm by 10 cm. The reason why we decided to go with a 10 by 10 QR code is because after testing with sizes ranging from 2 cm to 15 cm, the robustness of the system is compromised if we drop below that size. We saw satisfactory results with sizes down to 8 cm so a 10 cm QR code was deemed the best middle ground (Figure 5.1).

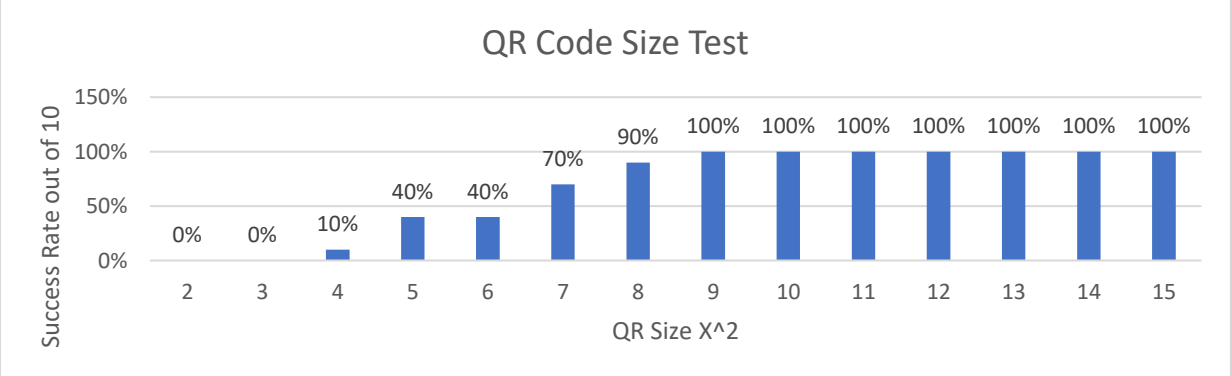

Figure 5.1 QR Code size test

The fixture in our case, will be specifically designed to attach on a DJI Matrice M210 (Figure 5.2). The reason we chose this specific drone is that it is available to us but also because it is close to the maximum payload capacity of the robotic arm therefore further testing its robustness.

Figure 5.2 DJI Matrice M210

In general or system will be able to handle any drone up to 3.5 kilograms and big enough to fit in the cameras area of view, as calculated in 4.2.2. Table 5.1 shows the maximum specifications that the system can handle reliably. By reliably, we mean that, regardless the orientation, the drone will be detected. That means that the limiting factor is vertical space in the area of view since it is smaller. Maximum height is defined by the distance of the camera from the landing pad after subtracting the cameras distance from the tip of the end effector. Also, it must be noted that the fixture must be the tallest object on the drone to avoid collisions with the end effector.

Table 5.1 Systems maximum capacity

Parameter	**Max Value**
Weight	3,5 kg
Height	55 cm
Width/Length	90 cm

The gripping fixture will have to be as light as possible, while maintaining structural integrity. It should impede the drone's ability to fly as little as possible.

Next step was to take our measurements. The DJI drone has 4 threaded holes on the main frame measuring 120 mm from one another in a square pattern. The xArm Gripper has 40 mm gripping pads. Therefore, our gripping fixture should attach to those points and provide an extrusion of at least 40 mm for the gripper to grip. We will also give a 5 mm overhang in order to achieve a more reliable fixation without the risk of slipping. The above measurements resulted in the below prints (Figure 5.3).

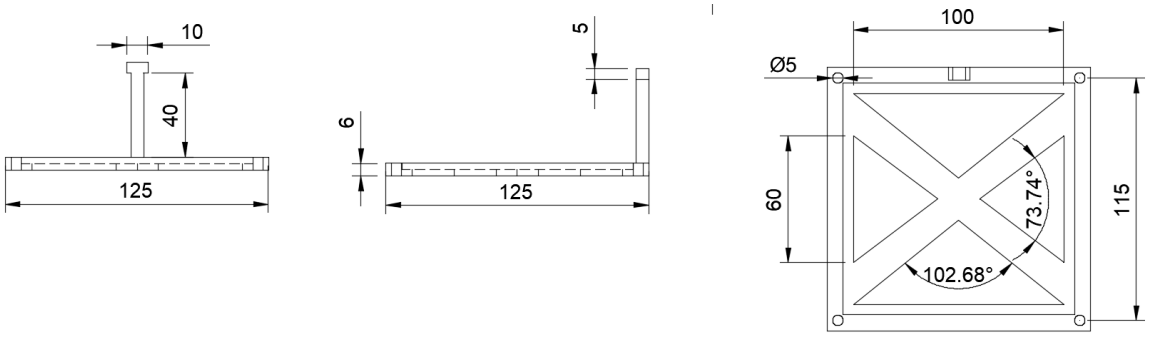

Figure 5.3 Fixture Prints

In Fusion 360, we created a flat surface that is sized 125x125 mm and 6cm thickness. We then created the four 2 mm holes in a square pattern of 120x120 mm. The extrusion was added with the overhead and we removed material from the base in order to reduce weight. In the end we were left with a satisfying prototype (Figure 5.4) and we are ready to export the STL file and proceed to 3d print the fixture.

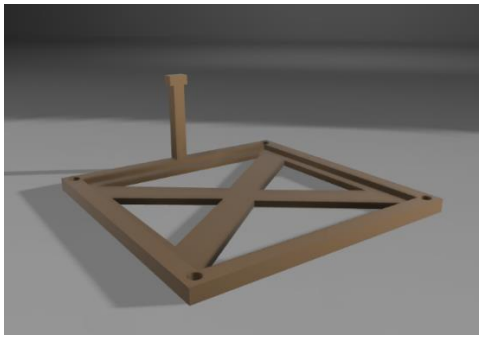

Figure 5.4 Fixture Final 3D Render

5.2. Implementation

Unfortunately, during the time of the implementation of our fixture, a global halting of shipments was applied not allowing us to the materials needed for the 3D Printing of our fixture. We were expecting 3D printing filament in order to utilize our 3D printer and print the fixture designed in the above chapter but unfortunately, we were not able to secure the filament and local warehouses were out of stock. That did not affect the progress of our thesis since we could machine the parts from older 3D printed scraps. We will continue with testing, using the test fixture as seen in Figure 4.6. We will still be printing new QR codes in order to compare them with the hand drawn one for the purpose of testing the systems robustness in the next chapter.

The test fixture worked as intended after a test run. It was able to withstand the force of the robotic gripper. It also had available protrusions in order to mount weights on it. We did this to test the capacity of the robotic arm and to simulate the weight of a real drone (Figure 5.5).

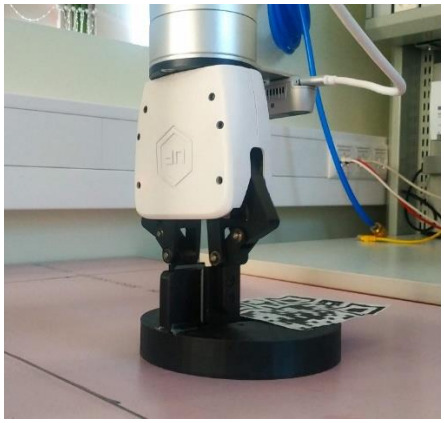

Figure 5.5 Gripping of the Test Fixture

6. FINAL IMPLEMENTATION

For the final Implementation we combine everything that we learned and all the solutions we found in the previous chapters. The process flow of our system is seen in Figure 6.1. This chart shows all the states that our system will possibly be in.

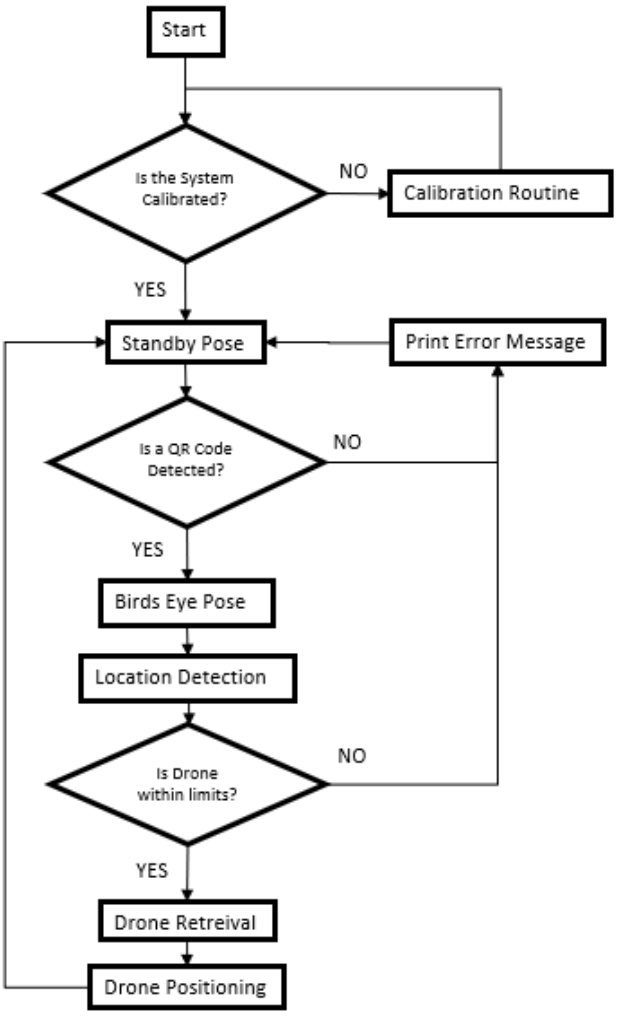

Figure 6.1 Process Flow Chart

6.1. Hardware

For the final implementation, we positioned the robotic arm, with the camera mounted on it, in front of the landing pad and rigidly connected them together. The USB type-C cable was attached to the robot in order to avoid it getting caught during the arms operation. Out test fixture was fitted with a QR code as seen in Figure 4.6.

The computer hardware was a commercially available laptop with no significant specification and the connection with the camera was established with a USB port. The xArm robot was connected through the uFactory motion controller to the computer though an Ethernet interface. The motion controller and computer were powered by 220V wall power while the camera was powered through the USB cable.

6.2. Software

On the software side, we combined every piece of code in one master Python file (A1 System Code). We included the needed libraries. Then we proceeded to initialize the robotic arm. This initialization includes the connection with it as well as the connection with the gripper. Then, we included the re-map function so that it can be used later in the code. Next step was to include all our local variables including our limits that were acquired by our calibration algorithm. At this point we also created a variable for the speed of our joints with will that will be used throughout the code.

Table 6.1 List of Variables

Name	**Type**	**Description**
sLoc	List	Storage Location Position
standby	List	Standby Position
birdsEye	List	Birds Eye position
camL,R,T,B	Integers	Limits of landing pad
xArmL,R,T,B	Integers	Limits of Robotic EE
speed	Integer	Global joint speed

The xArm API provides us with a specific set of commands in order to move the robotic arm. We can either define the angle of each of the 7 joints or the position of the end effector. In other words, we can use forward or inverse kinematics. In our case, in order to set the standby position and the Birds Eye position, we will use the angle method(6.1). That is why we defined those positions as lists. In order to move the end effector over the test picture, we will be using inverse kinematics. This means that we will input the position

calculated by our algorithm, in the command, in order to move the end effector at our desired location(6.2).

$$arm.set\_servo\_angle(angle = standBy, speed = speed, wait = True) \quad (6.1)$$

$$arm.set\_position(xPos, yPos, Depth, speed = speed, wait = True) \quad (6.2)$$

Next, in the code, we gave the command so that the robotic arm will proceed to the standby pose and the gripper at an open position. In our Prosses Flow Chart (Figure 6.1), this is the "Standby Pose" position. Since now everything is in place, we can activate our video stream and store the video frames in a variable after initializing our stream using the OpenCV library. Since there are no other cameras attached to our computer, the Realsence cameras identification code is "0". We are now in the standby state. As mentioned before, in the standby mode, the camera is able to overlook the landing pad, without physically obstructing it.

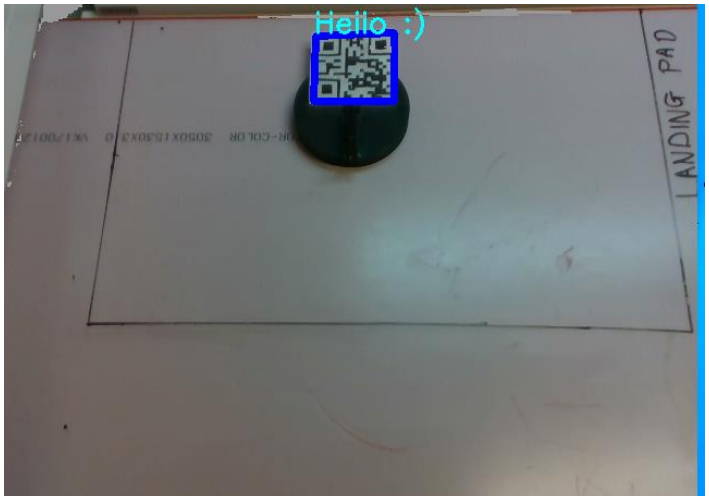

Figure 6.2 Camera stream from Standby pose

Next step is to start the main routine. Inside a while true statement, we will insert our detection code starting by passing the frames from our video stream into a variable called

cap. Then, using the decode function of the PYZBAR library, we will detect the presence of a QR code in each frame. We will pull each individual frame by storing it into a variable called *img*. When a QR code is not present, the decode function returns false therefore this can be used to detect the presence and activate our position detection algorithm. This way we can ensure that the system will not trigger in the presence of anything but a drone with a QR code on it. Consequently, nothing on the landing pad will trigger the system, ensuring that physical objects, like animals or leaves or shadows, will not trigger detection.

After a QR code is detected we set the robot in the Birds Eye pose and we can use our functions, described in 4.2.2, in order to get its location and angle. In our Prosses Flow Chart (Figure 6.1), this is the "Birds Eye" position. After we have calculated the position and angle, we use an if statement in order to check if the QR code is within the pre-determined limits. Those limits are set in our variable list and are found through the calibration process described in 3.2.2.

Figure 6.3 RGB Feed when in Bird Eye pose

After we have determined that the QR code is in a valid position we proceed using inverse kinematics to position the gripper over the fixture and then close the gripper. Then we will set the arm back to the Birds Eye position in order to avoid any collisions with a possible second drone that has landed on the landing pad. In our Prosses Flow Chart (Figure 6.1), this is the "Location Detection" position.

After we have ensured that we will not have a collision, we can proceed to place the fixture in or pre-defined storage location. In our Prosses Flow Chart (Figure 6.1), this is the "Drone Retrieval" and "Drone Positioning" position. The robot then will perform a clearing maneuver, so bad it won't bump into the test fixture. It will then return to its standby position waiting for a new QR code, returning to the beginning of the flow chart.

In case the QR code is not within limits, an error message will be printed. In case a QR code is not present, the system will also print a message Indicating that a QR code was not detected.

This final implementation shows us that our system has four basic states as indicated by the below state machine. Individual states are as follows: stand by, Birds Eye, Detection and Retrieval.

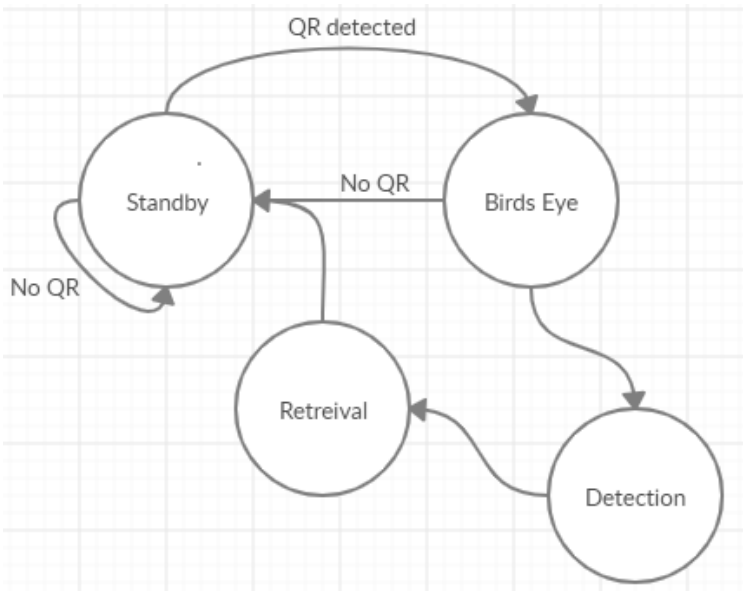

Figure 6.4 Systems State Machine

As we can see the only thing that can trigger the system to start detecting is the presence of a QR code. Even if a QR code is detected in standby mode, the system will still have to validate its presence while in the Birds Eye view. If the QR code has landed outside of our limits or for some reason has disappeared from the platform until the robot is at bird's eye view, the robot will return to its standby position.

7. TESTING

For testing purposes, we will split our focus into two categories. First category of testing will be investigating the accuracy of the system. This will measure how accurate our system is in detecting the drone and its position as well as how accurate the system places the drone on our desired location. The second category will be reliability testing. The scope of the reliability testing will be how accurately the system detects the drone depending on lighting conditions that will simulate real life operation. Another area for reliability testing will be to confirm the robustness of the system depending on the quality of the QR code itself and the lighting situations.

7.1. Testing Methodology

In order to define the system's accuracy, we used our test fixture to execute a detection and positioning of our test piece 50 times in a row. This testing will be conducted under several lighting conditions. The lighting conditions are as follows: LED illumination, under CFL illumination, and under sunlight. For each lighting condition, we will measure how accurately the system identified the drone, how accurately it translated its coordinates to the robotic arm positioning, and how accurate was its placement to our desired location.

According to X. Lu [32] the effective charging distance, of a wireless charger is generally within 20cm Therefore, we will define a center point with a 20 centimeter circle around it and record how many times the system was able to successfully place the test fixture within our defined limits. In this way we will both check for the accuracy of the positioning as well as if the system is a viable solution for a wireless charging application. This, in turn, will provide us with a proof that a generic positioning system is possible to be implemented for wireless charging use.

In order to test reliability, we will be performing a series of dynamic tests where the lighting conditions and the quality of the QR codes used will be dynamically changed within a set of detections. This will ensure that the system will be reliable and robust in detecting different quality QR codes as well as perform in dynamically changing environment. We will test for day and night cycles under different types of illumination, as described above.

In short, we will run 50 tests under a single elimination set, record the performance and then change the elimination scenario. We will then change the illumination type and repeat (Figure 7.1).

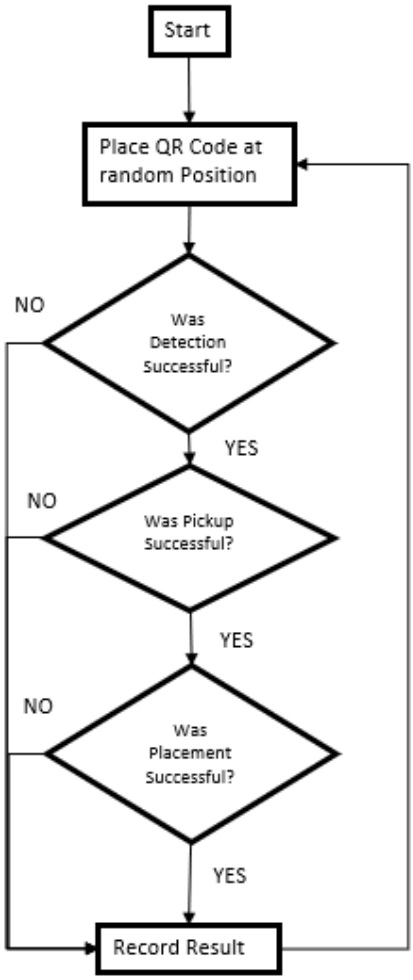

Figure 7.1 Testing Procedure

7.2. Testing Results

After we run all the tests and collected the data in the table seen in appendix, we reached the conclusion that the simplicity of our system is enough to provide a reliable solution. We saw high success rates under direct sunlight and under LED light. Under CFL light the robot would not detect as accurately as in other conditions. That is due to the nature of CFL light. It is known that CFL light emits waveforms that peak in certain frequencies that could pose a threat to the detection of our QR code. We also see that the placing is accurate since the system never failed to place the test fixture within limits.

Table 7.1 Test Results with High Quality QR Code

High quality QR	**CFL**	**LED**	**Daylight**	**Darkness**
Detection	41/50	48/50	47/50	0/50
Pickup	40/50	45/50	45/50	0/50
Placing	40/50	45/50	45/50	0/50
Success Rate	80%	90%	90%	0%

We also saw that the quality of the QR code does not affect the performance of the system. The true enemy of performance is darkness since the camera retrieves position in 2D space with the use of the RGB sensor. When there is no light it cannot collect light and therefor the information for the QR code is lost. As a result, darkness is rendering our system incapable of performing.

Table 7.2 Test Results with a Low Quality QR code

Low quality QR	**CFL**	**LED**	**Daylight**	**Darkness**
Detection	40/50	46/50	45/50	0/50
Pickup	40/50	45/50	45/50	0/50
Placing	40/50	45/50	45/50	0/50
Success Rate	80%	90%	90%	0%

Again, we see in the results, that the success rate of the placement itself was 100% therefore proving that our positioning is accurate.

It is possible to change the whole basis of the system and make it purely detect depth since darkness does not affect the depth sensor. In a future iteration this might be a good solution, in order to break away from the need of illuminating landing platforms.

8. Future Work

In this thesis we addressed the problem of accurate positioning of a drone post landing. Many different aspects of this solution have been left for the future due to the lack of time.

There are many ideas that I would have liked to test such as applying a machine learning algorithm in order to detect a drone without the use of a QR code. Also, this system could be expanded to accommodate a security system that will use the QR code in order to identify it. This could be especially useful as a commercial product. For example, if a company has a fleet of drones in order to perform deliveries of sensitive items and there is a need to secure the landing pad and discard any foreign drones.

Another aspect that I would like to touch, in the future, is the ability to use the depth sensor, solely, in order to perform all operations. This means that no RGB stream will be used and therefore the system will be able to perform in the dark as well. Then, machine learning could also be applied to the depth field and thus achieving a system that would not need any illumination.

Finally, the biggest improvement to this system will be to be able to detect and grasp a drone while it is hovering mid-air. this will open all new possibilities 4 drone detection and position systems. It will allow drones to not require to land in order to be retrieved, allowing for the possibility of moving landing vehicles.

9. SUMMARY

9.1. Summary (English)

In this thesis we managed to design and implement a drone detection and positioning system designed for a 7DOF robotic arm. A robotic arm was placed next to a landing pad and the system will respond when a drone has landed on the landing pad. The detection is possible with the use of machine vision, through an Intel Realsense camera.

The implemented system can detect a drone in 3D space, using an QR code placed on top of it. The systems RGB camera detects the QR code in the XY plane and the cameras depth sensor detects the Z position. It is then able to translate its detected position into a physical movement of the robotic arm with the use of a combination of forward and inverse kinematics.

The connection between the machine vision and the physical world is done by creating a linear dependency between the position of the QR code and the robotic arms end effector position in physical space.

This thesis was constructed with simplicity in mind and was able to solve machine vision problems, such as perspective distortion, by working around them allowing for a simple solution with low cost of deployment and maintenance. We also ensured that the system is robust and that it will not trigger in the event of a foreign object being present on the landing pad.

The resulting system was able to perform under most lighting conditions. The system did not perform in darkness due to the fact that it relies on the visible spectrum in order to detect the X and Y position of the drone.

Several improvements and expansions could be further developed such as a security system that will detect the presence of drones that are not supposed to be retrieved by the system. A development of a GUI could also prove useful for commercial application of the system.

9.2. Summary (Estonian)

Selles lõputöös õnnestus meil välja töötada ja rakendada droonide tuvastamise ja positsioneerimissüsteem, mis on mõeldud 7DOF-i robotkäe jaoks. Maandumispadja kõrvale asetati robotkäsi ja süsteem reageerib siis, kui droon on maandumispadjale maandunud. Avastamine on võimalik masinnägemise abil Inteli Realsense kaamera kaudu.

Rakendatud süsteem suudab 3D-ruumis drooni tuvastada, kasutades selle peale pandud QR-koodi. Süsteemide RGB-kaamera tuvastab QR-koodi tasapinnal XY ja kaamerate sügavuse andur tuvastab Z-positsiooni. Seejärel on see võimeline tõlgendama oma tuvastatud asendi robotkäe füüsiliseks liikumiseks, kasutades selleks kinemaatika edasi-tagasi pöördeid.

Seos masinnägemise ja füüsilise maailma vahel luuakse lineaarse sõltuvuse abil QR-koodi asukoha ja robotrelvade otsaefektori positsiooni vahel füüsilises ruumis.

See lõputöö oli koostatud lihtsust silmas pidades ja suutis uut lähenemist pakkudes lahendada masinnägemisprobleeme, näiteks perspektiivi moonutusi, võimaldades efektiivse lahenduse madalate juurutamise ja hoolduskuludega. Samuti veendusime, et süsteem on vastupidav ja et see ei käivitu, kui maandumispadjal on võõrkeha.

Saadud süsteem oli võimeline toimima enamikes valgustingimustes. Süsteem ei töötanud pimeduses seetõttu, et drooni X- ja Y-positsiooni tuvastamiseks tugineb see nähtavale spektrile.

Edaspidi võiks välja töötada mitmeid parandusi ja laiendeid, näiteks turvasüsteemi, mis tuvastab droonide olemasolu, mida süsteem ei peaks väidetavalt hankima. GUI arendamine võib osutuda kasulikuks ka süsteemi kommertsrakenduste jaoks.

10. Bibliography

[1] f. Staff, "On This Day: Austria Drops Balloon Bombs on Venice," 22 August 2011. [Online]. Available: http://www.findingdulcinea.com/news/on-this-day/July-August-08/On-this-Day--Austria-Rains-Balloon-Bombs-on-Venice.html. [Accessed 20 March 2020].

[2] X. Huang and X. Hu, "Orthogonal design and optimization of flight stability test for the quadrotor unmanned aerial vehicle," *2017 IEEE International Conference on Unmanned Systems (ICUS) Beijing,* pp. 343-346, 2017.

[3] T. Costa, M. Caldas, P. Araujo and E. Silva, "Topological Optimization applied towards the development of a small and lightweight MAV composite frame," 2018.

[4] DJI, "DJI," 2020. [Online]. Available: https://www.dji.com/ee/mavic-air. [Accessed 2020].

[5] N. Mathew, S. L. Smith and S. L. Waslander, "Planning Paths for Package Delivery in Heterogeneous Multirobot Teams," *IEEE Transactions on Automation Science and Engineering,* vol. 12, no. 4, pp. pp. 1298-1308, 2015.

[6] K. Dorling, J. Heinrichs, G. G. Messier and S. Magierowsk, "Vehicle Routing Problems for Drone Delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 47, no. 1, pp. 70-85, 2017.

[7] S. Tansuriyong, M. Kyan, K. Numata, S. Taira and T. Anezaki, "Verification experiment for drone charging station using RTK-GPS," in *2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Okinawa, 2017.

[8] uFactory, "xArm," 2020. [Online]. Available: https://www.xarm.cc/. [Accessed 2020].

[9] C. Guide, "UNIVERSAL ROBOTS | UR3, UR5, UR10," 2017. [Online]. Available: https://cobotsguide.com/2016/06/universal-robots/. [Accessed 2020].

[10] D. ADC, "http://www.nacs.org/," 12 May 2011. [Online]. Available: http://www.nacs.org/LinkClick.aspx?fileticket=D1FpVAvvJuo%3D&tabid=1426&mid=4802. [Accessed 2 May 2020].

[11] D. Martin, "A practical guide to machine vision lighting," *Adv Illum,* pp. 1-3, 2007.

[12] L. Burke, "The Ra Optics Guide to Artificial Lighting," 19 June 2019. [Online]. Available: https://raoptics.com/blogs/news/the-ra-optics-guide-to-artificial-lighting. [Accessed 20 March 2020].

[13] C. Burke, H. Nguyen, M. Magilligan and R. Noorani, "Study of A Drone's Payload Delivery Capabilities Utilizing Rotational Movement," in *2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)*, Dhaka, 2019.

[14] Amazon, "Amazon Prime Air," 2020. [Online]. Available: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011. [Accessed 20 March 2020].

[15] D. Shneider, "The delivery frones are coming," *IEEE Spectrum,* vol. 1, no. 57, pp. 28-29, 2020.

[16] Jiang, T. Zhao and Hong, "Landing system for AR.Drone 2.0 using onboard camera and ROS," in *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, Nanjing, 2016.

[17] B. Shoufan and A. Taha, "Machine Learning-Based Drone Detection and Classification: State-of-the-Art in Research," *IEEE Access,* vol. 7, no. 1, pp. pp. 138669-138682, 2019.

[18] Z. Li, H. Liu and D. Sun, "Moving object detection and locating based on region shrinking algorithm," in *2012 IEEE International Conference on Mechatronics and Automation*, Chengdu, 2012.

[19] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler and R. Urtasun, "3D Object Proposals Using Stereo Imagery for Accurate Object Class Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 40, no. 5, pp. 1259-1272, 2018.

[20] G. Luo, M. Cheng and C. Chiang, "Vision-based 3-D object pick-and-place tasks of industrial manipulator," in *2017 International Automatic Control Conference (CACS)*, Pingtung, 2017.

[21] J. Liu, H. Chen and J. Li, "Faster 3D Object Detection in RGB-D Image Using 3D Selective Search and Object Pruning," in *2018 Chinese Control And Decision Conference (CCDC)*, Shenyang, 2018.

[22] N. A. Abramov, M. V. Bolsunovskaya, A. V. Leksashow and D. S. Barinov, "Algorithms for detecting and tracking of objects with optical markers in 3D space," in *2016 XIX IEEE International Conference on Soft Computing and Measurements (SCM)*, St. Petersburg, 2016.

[23] R. Yenorkar and U. M. Chaskar, "GUI Based Pick and Place Robotic Arm for Intelligent Computing and Control Systems," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, 2018.

[24] M. Farag, A. N. A. Ghafar and M. H. Alsibai, "Grasping and Positioning Tasks for Selective Compliant Articulated Robotic Arm Using Object Detection and Localization: Preliminary Results," in *2019 6th International Conference on Electrical and Electronics Engineering (ICEEE)*, Istanbul, 2019.

[25] R. Kumar, S. K. S. Lal and P. Chand, "Object detection and recognition for a pick and place Robot," in *Asia-Pacific World Congress on Computer Science and Engineering*, Nadi, 2014.

[26] S. A. Khan, T. Z. Anika, N. Sultana, F. Hossain and M. N. Uddin, "Color Sorting Robotic Arm," in *2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)*, Dhaka, 2019.

[27] R. S. a. A. Gontean, "Controlling a robotic arm in the 3D space with stereo vision," in *21st Telecommunications Forum Telfor (TELFOR)*, Belgrade, 2013.

[28] P. Andhare and S. Rawat, "Pick and place industrial robot controller with computer vision," in *2016 International Conference on Computing Communication Control and automation*, Pune, 2016.

[29] uFactory, "xArm Camera Module," 2020. [Online]. Available: https://store.ufactory.cc/products/xarm-camera-module-2020. [Accessed 20 March 2020].

[30] xArm, "xArm Gripper -2019," 2019. [Online]. Available: https://store.ufactory.cc/products/xarm-gripper. [Accessed 20 March 2020].

[31] Intel, "Intel® RealSense™ Depth Camera D400-Series Datasheet," 2019. [Online]. Available: https://www.mouser.com/pdfdocs/Intel_D400_Series_Datasheet.pdf. [Accessed 20 March 2020].

[32] X. Lu, P. Wang, D. Niyato, D. I. Kim and Z. Han, "Wireless Charging Technologies: Fundamentals, Standards, and Network Applications," *IEEE Communications Surveys & Tutorials,* vol. 18, no. 2, pp. 1413-1452, 2016.

[33] R. Gontean and A. Szabó, "Controlling a robotic arm in the 3D space with stereo vision," in *21st Telecommunications Forum Telfor (TELFOR)*, Belgrade, 2013.

[34] A. W. SUDBURY and E. B. HUTCHINSON, "A Cost Analysis Of Amazon Prime Air (Drone Delivery)," *Journal for Economic Educators,* vol. 16, no. 1, pp. 1-12, 2020.

List of Figures

List of Tables

A Appendix A

A1 System Code

```
# Imports
import os
import sys
import time
import math
import numpy as np
import cv2
from pyzbar.pyzbar import decode
import pyrealsense2 as rs

sys.path.append(os.path.join(os.path.dirname(__file__), '../../..'))
from xarm.wrapper import XArmAPI
from configparser import ConfigParser

pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.depth, 1280, 720, rs.format.z16, 6)
pipeline.start(config)

# Connect to xArm
parser = ConfigParser()
parser.read('C:\\Users\\geo_t\\PycharmProjects\\xArm\\xarm\\wrapper\\robot.conf')
ip = parser.get('xArm', 'ip')

# xArm init
arm = XArmAPI(ip)
arm.motion_enable(enable=True)
arm.set_mode(0)
arm.set_state(state=0)

# Gripper init
arm.set_gripper_mode(0)
arm.set_gripper_enable(True)
arm.set_gripper_speed(5000)


# Interpolation function
def remap(x, oMin, oMax, nMin, nMax):
    # range check
    if oMin == oMax:
        print("Warning: Zero input range")
        return None

    if nMin == nMax:
        print("Warning: Zero output range")
        return None

    # check reversed input range
    reverseInput = False
    oldMin = min(oMin, oMax)
    oldMax = max(oMin, oMax)
    if not oldMin == oMin:
        reverseInput = True

    # check reversed output range
    reverseOutput = False
    newMin = min(nMin, nMax)
    newMax = max(nMin, nMax)
    if not newMin == nMin:
```

```
        reverseOutput = True

    portion = (x - oldMin) * (newMax - newMin) / (oldMax - oldMin)
    if reverseInput:
        portion = (oldMax - x) * (newMax - newMin) / (oldMax - oldMin)

    result = portion + newMin
    if reverseOutput:
        result = newMax - portion

    return result

# Variables
birdEye = [0.0, 3.3000077168354895, 0.0, 116.00000387815524, 4.119222772313541, 113.93036573058367,
           0.0]  # Default Birdseye view
# birdEye =[429.121399, 8.58317, 661.836731, -3.075262, 0.020507, 0.029147]#birdeye position
sLoc = [-65.2, 27.9, 0, 56.1, -0.2, 27.8, -65]
sLocRes = [0, 7, 0, 107, 0, 100, 0]
standBy = [0, -65, 0, 69, 0, 113.4, 0]
maxHeight = 541.3
# camera area calibration DLT
camL = 390
camR = 1127
camLR = [camL, camR]
camT = 14
camB = 319
camTB = [camT, camB]

# xArm reach calibration
xArmL = 240
xArmR = -303
xArmLR = [xArmL, xArmR]
xArmT = 591
xArmB = 361
xArmTB = [xArmT, xArmB]

# speed of joints
speed = 50

# robo position INIT
      # Goto default Birdseye view
      arm.set_servo_angle(angle=standBy, speed=speed, wait=True)
      # Default gripper opening
      arm.set_gripper_position(800, wait=True)

      # capture from Camera
      cap = cv2.VideoCapture(0)
      cap.set(3, 1280)
      cap.set(4, 720)

      while True:
          frames = pipeline.wait_for_frames()
          depth = frames.get_depth_frame()
          success, img = cap.read()

          arm.set_servo_angle(angle=standBy, speed=speed, wait=True)
          arm.set_state(state=0)
          for barcode in decode(img):
              arm.set_servo_angle(angle=birdEye, speed=speed, wait=True)
              arm.set_state(state=0)
              time.sleep(2)
              poly = barcode.polygon  # .polygon creates an array with coordinates of each corner of the qr code
```

```
                # Take the 2 bottom corners
                xy2 = poly[2]
                x2 = int(xy2[0])
                y2 = int(xy2[1])

                xy3 = poly[3]
                x3 = int(xy3[0])
                y3 = int(xy3[1])

                # get angle of bottom
                radians = math.atan2(y2 - y3, x2 - x3)
                degrees = round(math.degrees(radians))

                # calculate the midpoint between the two bottom points
                xPos = int((x3 + x2) / 2)
                yPos = int((y3 + y2) / 2)
                xLoc = round(remap(yPos, camB, camT, xArmB, xArmT))
                yLoc = round(remap(xPos, camR, camL, xArmR, xArmL))
                print(xLoc, yLoc)
                print(degrees)
                dp = (maxHeight-depth.get_distance(xPos, yPos))
                print(dp)

                if xArmB <= xLoc <= xArmT and xArmR <= yLoc <= xArmL:
                    # Reset Position to avoid singularities
                    arm.set_servo_angle(angle=sLocRes, speed=speed, wait=True)
                    # Input angle of qr
                    arm.set_servo_angle(angle=[sLocRes[0], sLocRes[1], sLocRes[2], s
    LocRes[3], sLocRes[4], sLocRes[5], degrees], speed=speed, wait=True)
                    # Go to position of QR
                    arm.set_position(xLoc, yLoc, dp, 0, 0, 0, wait=True)
                    # Grab fixture
                    arm.set_gripper_position(60, wait=True)
                    # return to bird eye position
                    arm.set_servo_angle(angle=birdEye, speed=speed, wait=True)
                    # move to storage location
                    arm.set_servo_angle(angle=sLoc, speed=speed, wait=True)
                    # Release
                    arm.set_gripper_position(800, wait=True)
                    # retract arm
                    arm.set_servo_angle(servo_id=2, angle=-
    40, speed=speed, wait=True)
                    # Return to BirdEye before going standby
                    arm.set_servo_angle(angle=birdEye, speed=speed, wait=True)
                    # Reset arm in case of failure
                    arm.set_state(state=0)
            else:
                print("No QR or QR out of bounds")

        arm.disconnect()
```

A2 Python Libraries Used

Table A.0.1 Libraries used

Name	**Description**
os	Operating system interfaces Library
sys	System-specific parameters and functions Library
time	Time handling Library
math	Mathematical functions Library
numpy	Matrix manipulation Library
cv2	OpenCV Library
pyzbar	QR detection Library
pyrealsense	Intel Realsense Camera Library