Distributed Architecture for Sensorial Implementation

Jorge Tiago da Rocha Afonso
tiago.afonso@tecnico.ulisboa.pt

Instituto Superior Tecnico, Universidade de Lisboa, Portugal

March 2019

Abstract

This paper proposes a complete system to support the development of advanced navigation and control algorithms within an indoor laboratory environment. A fully distributed architecture, named ADIS, is developed allowing for a flexible solution. Techniques for both 2D and 3D real-time marker position detection are presented based on machine vision techniques. A point tracking algorithm is included using Kalman filter estimation. The whole system was built using common hardware and sensors since the overall cost was a limiting factor.
As for vehicles, an AR.Drone 2.0 was used. In this paper, only flying vehicles are explored because the grounded vehicles were addressed in [1]. Consumer grade quadcopters, more commonly known as drones, were deeply customized on the software side to integrate seemly with the rest of the system. By analyzing its onboard operating system and control software, and by using several freely available open source software and tools, a close integration between system and vehicle was achieved to provide a user-friendly development platform.
Finally, a performance evaluation was done, with the intent of evaluating the reliability and overall behavior of the system. This evaluation was more focused on localization system precision and speed since it is the core functionality of the system.

1 Introduction

There is no doubt that scientific and technological advances have transformed our world and shaped our lives and the impact of technology in our world is undeniable.

Mobility has always been one of the areas of technology that changed our life the most. The invention of the automobile and its mass production ended an era where animal strength was the heart of the transportation industry and started a revolution where everyone could travel with no effort and transport goods efficiently. After the automobile, the new revolution on mobility was the airplane. It connected countries and reduced distances, making trips that used to last months to last just a few hours. Currently, the next step in mobility technology could be autonomous driving and navigation. Most vehicles still require at least one human driver, which represents not just an annoyance to the driver, but also a security risk. A complete autonomous vehicle could not only improve security and reduce crashes but also increase efficiency and speed in transportation. Autonomous or self-driving vehicles can also be used in many industrial applications like herbicide spraying, harvest cropping, automated package delivery, private security or even in fields where human lives are at risk like firefighting or military applications.

This kind of vehicles relies on multiple types of navigation and control algorithms to cruise, maintain stability and to make decisions. These algorithms need to be able to execute tasks like sense and avoid obstacles, optimize routes according to multiple factors such as energy efficiency or traffic, cooperate with other vehicles to share information like position and trajectory planning, prioritize tasks and actions to avoid imminent risks or to guarantee driving comfort, among others.

This paper is not going to focus on navigation and control algorithms but rather on creating tools and infrastructure, named ADIS, to support their development, laboratory implementation and testing. When the goal is real-world use of navigation, and control techniques, implementation, validation and refinement beyond simulation are essential to go further than the initial theoretical construct. A controlled laboratory environment becomes then the natural next step in the development process. However, this requires specialized equipment or infrastructure capable of accommodating multiple controller types to be deployed and its behavior measured. With this in mind, the emphasis of this thesis is to create a development platform, for laboratory implementation and testing of advanced control algorithms aimed at grounded and airborne vehicles.

2 System Overview

The ADIS system was made to provide a wide framework and toolset, for development and testing of navi-

gation algorithms. The basic requirements set for the final system were: access to reliable location data, access to telemetry vehicle data and easy to use tooling and interface.

The first requirement dictates the need to know the controlled vehicle location in space. The location coordinates are measured in a user-defined inertial frame, since navigating a vehicle through space is a prime objective for any navigation controller. Depending on the application either a two dimensional or a three-dimensional inertial frame may be needed.

The second requirement addresses the need to know the orientation and inertial state of the controlled vehicle. To gather this telemetry data, each vehicle must be equipped with a wide range of onboard motion sensors.

Finally, the third requirement is related to the ease of use of the final system. Because the ultimate goal of the ADIS system is to be used in the real world for academic development, it is imperative that it provides a simple and user-friendly workflow.

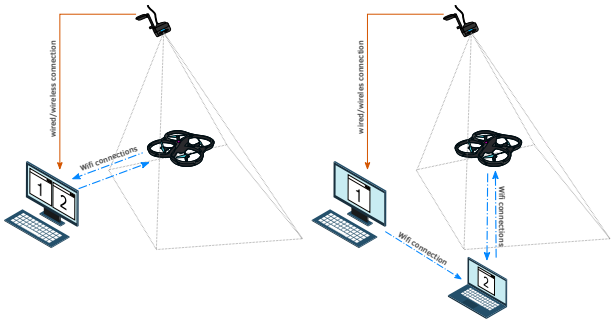
Taking these requirements as guidelines, the finalized system can be divided into three distinct parts with three distinct functions: a localization system to acquire location data, a vehicle integration layer to accommodate all vehicles and its onboard sensors, and a communication layer serving as the backbone for all system components interoperability, handling all communication needs.

2.1 System Architectures

The various components of the ADIS system can be interconnected in multiple ways. Multiple system architectures can be used, tailoring to the user's needs and level of expertise.

The simplest topologies that ADIS has are shown in figure 1. Only one vehicle is used and is connected to the control algorithm using a Wi-Fi network. The same Wi-Fi network is used for the drone to broadcast back its telemetry data. Because the main computer resources main not be enough to have the localization software and control algorithm running in the same machine (figure 1a) it is possible to use an external computer to run the control algorithm, which receives the location data through a wired Ethernet connection or through the already mentioned Wi-Fi network.

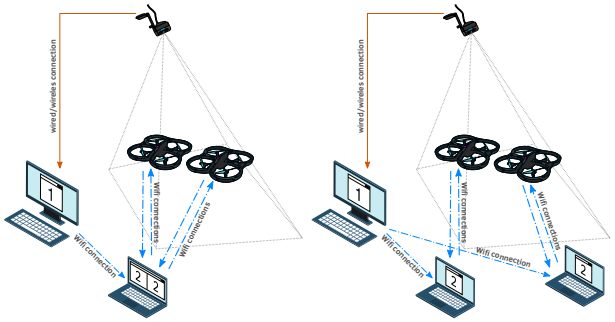
If multiple drones are required, it is possible to run multiple control algorithms in the main computer, however, it is recommended the use of one or more external computers like in figure 2a and figure 2b.



(a) All in one machine (b) More than one machine

1 - ADIS Software Suite
2 - User Controller

Figure 1: ADIS basic architectures



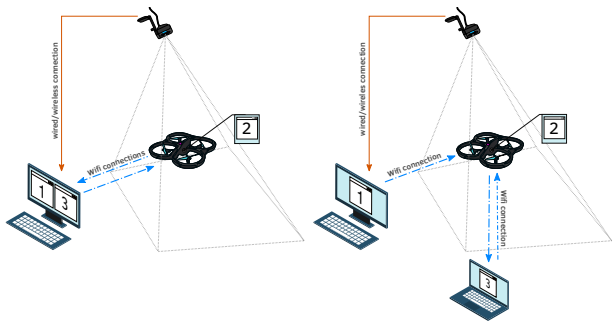
(a) External machine with multiple vehicles (b) Multiple external machines and multiple vehicles

1 - ADIS Software Suite
2 - User Controller

Figure 2: ADIS multi-vehicle architectures

All of the architectures previously described assume that the vehicle's control algorithm is running in an external computer. However, if a drone is used, it is possible to write and compile binaries that run directly inside the drone's onboard computer, as described in section 5. These compiled binaries offer a remote visualization and control interface which is executed in an external computer.

Figures 3a and 3b, describe two different architectures using an onboard controller, without or without the use of extra computers.



(a) External controller running inside the drone (b) Multiple external machines and multiple vehicles

1 - ADIS Software Suite
2 - User Controller
3 - Data Visualization

Figure 3: ADIS onboard algorithm architectures

In this architecture, the main computer is merely receiving and displaying a reduced set of data returned by the vehicle. Some control commands and flags can be set within this visualization interface, but the bulk of the control algorithm's work is relocated to the drone's onboard processing unit. Reduces the load on the main computer freeing resources for more vehicles and cameras to be added. This topology also changes the destination of the location coordinates sent by the localization system from the main computer to the drone.

3 Theoretical Overview

In optical localization technologies, one of the main tasks is to associate pixel coordinates, measured from a digital sensor, to real-world coordinates from a visible scene. This association should be described by a well defined mathematical model, which allows a computerized system to interpret the real world and understanding its surroundings.

3.1 Homogeneous Coordinates

Allow to represent a 2D point as 3D point by using a "fictitious" coordinate. By convention, from a point in homogeneous coordinates (x', y', z') is always possible to recover (x, y) using:

$$x = \frac{x'}{z'} \qquad y = \frac{y'}{z'} \tag{1}$$

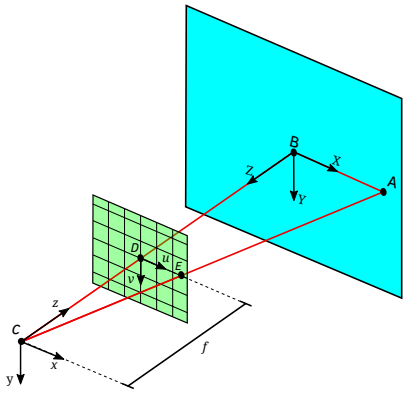
These coordinates are called homogeneous because the overall scaling of the coordinates is not important, because:

$$x = \frac{x'}{z'} = \frac{kx'}{kz'} \qquad y = \frac{y'}{z'} = \frac{ky'}{kz'}$$

This means that a representation of a Cartesian point (x, y) is not unique but independent of scaling.

3.2 Pinhole Model

The pinhole model describes how the three-dimensional coordinates of a point in space, are mathematically related to its projection onto the image plane of an ideal camera. The image plane corresponds to the plane in the world from which the scene is viewed, and in the case of a digital camera, it's the plane of the photosensitive sensor.



(a) Defining a triangle $\triangle ABC$

Figure 4: Pinhole Geometry

Figure 4 shows the pinhole camera geometry. Point C, marks the hole of the camera: here the light enters the opaque box, the blue surface corresponds to the visible area, the green plane is the captured image plane, distance f is called focal length and point B is an arbitrary point of interest in the visible area represented by point D in the acquired image. This geometry creates two similar triangles $\triangle ABC$ and $\triangle CDE$, which lead to

$$\frac{\overline{AB}}{\overline{CB}} = \frac{\overline{ED}}{\overline{DC}} \Leftrightarrow \overline{AB} = \overline{CB} \times \frac{\overline{ED}}{\overline{DC}}$$

So if point B is given by (X, Y) and point D by (u, v), then

$$X = \frac{Z \times u}{f} \qquad Y = \frac{Z \times v}{f} \tag{2}$$

3.3 World to Camera Transformation

To transform points from the inertial frame to the camera frame, a translation, T, and a rotation, R, are applied to the inertial frame, B.

$$A_c = RA_B + T \tag{3}$$

The parameters T and R are called extrinsic parameters.

3.4 Sensor Plane to Pixel Coordinates Transformation

To apply the pinhole model, the pixel array return by a digital camera must be converted to metric units. Each pixel of the image array is identified with a pair of indexes that do not correspond to the Cartesian coordinates measured in the sensor frame. As seen

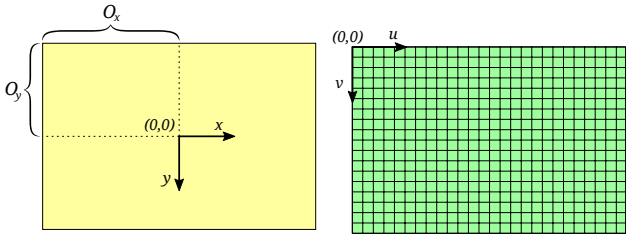


Figure 5: Sensor to Pixels

in figure 5, it's necessary to use offsets to match the origin point.

$$u = x + O_x \qquad v = y + O_y$$

To convert from metric units, u and v are divided by each cell of the sensor.

$$u = \frac{x}{s_x} + O_x \qquad v = \frac{y}{s_y} + O_y \tag{4}$$

3.5 Planar Homography

Using the equations 2, 3 and 4, with homogeneous coordinates we get:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} \frac{1}{s_x} & 0 & O_x \\ 0 & \frac{1}{s_y} & O_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{5}$$

after simplification, yields,

$$\begin{bmatrix} p \\ q \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}^{-1} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \tag{6}$$

Equation 6 is able to convert coordinates from the camera sensor plane, in pixels, to the coordinates in the inertial frame.

3.6 Projection of a point onto a plane

Consider the plane defined by a point $P_0 = (x_0, y_0, z_0)$ and a normal vector $\vec{n} = (a, b, c)$. The projection of an arbitrary point Q not contained in the plane is given by a point R so that the dot product between vectors $\overrightarrow{RP}$ and $\overrightarrow{RQ}$ is equal to zero. Then R is:

$$R = (x + ka\,,\, y + kb\,,\, z + kc)$$

making

$$\begin{aligned} \overrightarrow{RP} &= [-x + ka + x_0\,,\, -y + kb + y_0\,,\, -z + kc + z_0] \\ \overrightarrow{RQ} &= [-x + ka + x\,,\, -y + kb + y\,,\, -z + kc + z] \\ &= [ka\,,\, kb\,,\, kc] \end{aligned}$$

applying the dot product $0 = \overrightarrow{RP}.\overrightarrow{RQ}$ and solving for k,

$$k = \frac{-xa - yb - zc + x_0a + y_0b + z_0c}{a^2 + b^2 + c^2} \tag{7}$$

which allows to completely define the projection point R.

3.7 Singular Value Decomposition (SVD)

The singular value decomposition is a factorization of a real or complex matrix, with many applications in technological areas like in signal processing and statistics. Formally the singular value decomposition theorem states that any matrix $A \in \mathbb{R}^{m \times n}$, can be decomposed into two orthogonal matrices $U \in \mathbb{R}^{m \times m}$

, $V \in \mathbb{R}^{n \times n}$ and a diagonal matrix $\sum \in \mathbb{R}^{m \times n}$, i.e.,

$$\Sigma = \begin{bmatrix} \sigma_1 & & & & 0 & \cdots & 0 \\ & \ddots & & & & & \\ & & \sigma_r & & & & \\ & & & 0 & & & \\ & & & & \ddots & & \\ & & & & 0 & \cdots & 0 \end{bmatrix} \quad \text{for} \quad m \leq n$$

with diagonal entries

$$\sigma_1 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_{min}\{m, n\} = 0$$

such that

$$A = U\Sigma V^T$$

The diagonal entries σ_i, of Σ are called singular values of A. The columns of V are called right singular vectors and the columns of U are called left singular vectors.

This technique has many applications, one of them is the ability to solve plane fitting problems as described in "Solving Problems in Scientific Computing using Matlab and Maple", by W. Gander and J. Hrebicek[2].

4 Localization System

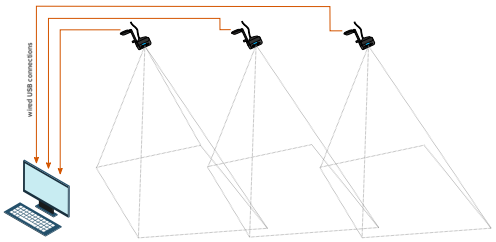
The localization system as both the option for two-dimensional or three-dimensional coordinates detection and tracking.

4.1 2D Localization

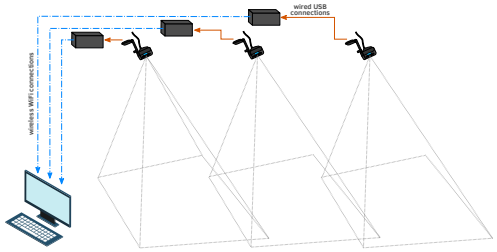
The 2D implementation of the localization system uses image processing to detect and track specialized markers, placed on points of interest defined by the user. The 2D version can be composed by multiple cameras wired directly to the main computer or by multiple cameras connected to the main computer wirelessly through Wi-Fi.

The basic operation concept consists of five different steps. First, digital cameras and machine vision detect visible markers. Then each detected marker is associated with others detected in previous frames. If no match is found, it's assumed that a new marker appeared. After that, the pixel coordinates of each marker is transformed into inertial frame coordinates using equation 6. Finally, the outgoing message with the data resulting from the previous steps is assembled and sent to the navigation controller.

Because equation 6, assumes that all visible points are on the same plane, it is not possible to use any



(a) Wired Architecture



(b) Wireless architecture

Figure 6: Localization system architectures

kind of flying vehicle. To go around this restriction, without using the 3D version of the tracking system, equation 8 can be used. If the vehicle in use is able to measure its height, the geometry from figure 7 allows for the following rationale.

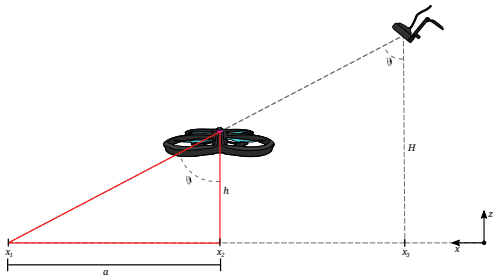


Figure 7: Coordinate correction for flying vehicles

If the drone is at height h, the localization system perceives its location as x_1 instead of x_2, then,

$$\begin{cases} x_1 - x_2 = a \\ tan\,\theta = \frac{a}{h} \\ tan\,\theta = \frac{x_1 - x_3}{H} \end{cases} \Leftrightarrow \quad x_2 = x_1 - \frac{(x_1 - x_3)h}{H}$$

the same is valid for the y axis, leading to:

$$x_2 = x_1 - \frac{(x_1 - x_3)h}{H} \qquad y_2 = y_1 - \frac{(y_1 - y_3)h}{H} \tag{8}$$

However, this solution has problems: the camera

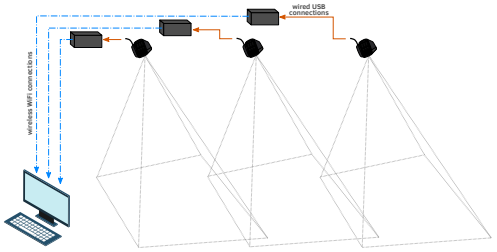


Figure 8: Distributed Architecture for the 3D system

height H and position (x_3, y_3) needs to be measured manually, and it requires the localization system to establish a connection with each drone to access its height. The first problem obviously forces the user to measure each camera height which are typically located in the ceiling and its exact location in the inertial frame. The second problem forces the localization system to associate which drone to one marker or a group of markers, adding more processing load and increases the overall complexity with more connections. To circumvent the second problem, equation 8 needs to be implemented on the navigation controller side. Each controller already has a connection established with its corresponding drone, making it possible to access the drone height directly and correct the location coordinates.

4.2 3D Localization

The 3D version of the localization system uses multiple Kinect v1 sensors wired directly to remote processing units that send the location results back to the main computer running the localization system software.

Although other depth-sensing cameras are commercially available like the Intel RealSense cameras or the Persee camera from Orbbec, a Kinect v1 sensor from Microsoft was used. The Kinect v1 sensor shipped with the Xbox 360 gaming console [3], and comes equipped with two cameras. One that captures RGB video, and another infrared camera that captures depth information. The equipped depth sensor, captures the distance from the camera plane, to the object. Similarly to the 2D version, the 3D localization system goes through the same operational phases: marker detection, coordinates transformation, marker association/tracking, message assembly, and the sending process. However, there are some differences during the conversion from pixel coordinates to real-world coordinates, due to the extra depth data. This time, the planar homography model given by the equation 6, is not enough to compute the height of an airborne vehicle.

To compute the correct inertial frame coordinates taking height into account, first, an estimation of the ground plane is calculated. This creates a reference plane from which all height measurements are calculated from, minimizing the errors caused by the depth sensor itself. If the floor is a perfect plane, it is geometrically represented by:

$$Ax + By + Cz + D = 0 \tag{9}$$

To estimate the ground plane, the iterative Random Sample Consensus algorithm, also known as RANSAC [4], is used. The RANSAC algorithm outputs a good estimation for the parameters A, B, C and D. However, these parameters define a planar equation using a reference frame with its origin located in the center of the Kinect and not in the inertial frame origin.

A good estimation of the floor position allows to transform the depth information into 3D location coordinates. The geometric construct from figure 9,

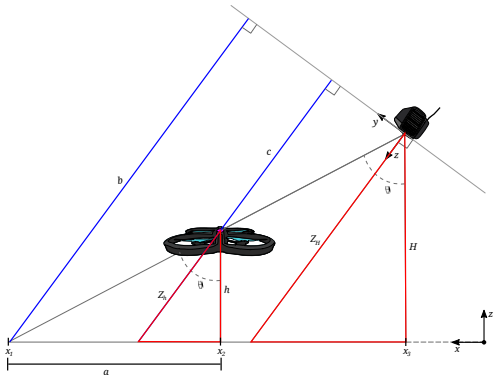


Figure 9: Geometric construct for height calculation

exposes two triangles (in red) that allow to calculate the height of the drone relative to the ground. Applying triangles similarity to the large and small triangles $\triangle_1 \sim \triangle_2$,

$$\frac{H}{Z_H} = \frac{h}{Z_h} \Leftrightarrow h = \frac{(b-c) \times H}{Z_H} \tag{10}$$

were h is the vehicle height, H is the camera height, c, b, and Z_H are distances perpendicular to the camera plane, between the camera plane and the marker, between the camera plane and the apparent vehicle location, and from the camera to the ground respectively. These quantities can all be determined using the ground plane estimation previously estimated. Equation 10, provides a way to calculate the vehicle's height directly, without the use of an onboard

altimeter. Still, it remains the problem that, due to its height, the drone is detected as a marker located in a point further away than it actually is. Equation 8, corrects these coordinates, however, it requires quantities that need to be measured manually for each camera. This is not a viable implementation, because every time the position of any Kinect in the system changes, new measurements are required. To avoid this, it's possible to calculate the pixel coordinates that the point defined by the marker would have, if it was sitting on the ground and then repurpose the 2D version strategy of using one homography matrix per camera to convert from pixel coordinates to initial frame coordinates, ending up with the coordinates corrected for the vehicle's height. In other words, we need to find the projection of the point onto the estimated ground plane and convert it to pixels and then calculate its coordinates in the initial frame of reference using homography. Looking at figure 9, the abscissa of this projected point is represented by x_2. To calculate this, we use the mathematical method described in section 3.6, using the parameters A, B, and C generated with the RANSAC algorithm. Assuming the arbitrary point in the ground plane P_0, as $P_0 = (0, 0, z_0)$, where z is given by the estimated ground plane definition,

$$0 = A \times 0 + B \times 0 + C \times z_0 + D$$
$$z_0 = -\frac{D}{C}$$

and the normal vector $\vec{n}$ is given by $\vec{n} = [A, B, C]$. So if the detected marker point is $Q = (x, y, z)$, the projected point onto the ground plane is

$$R = (x + kA, y + kB, z + kC)$$

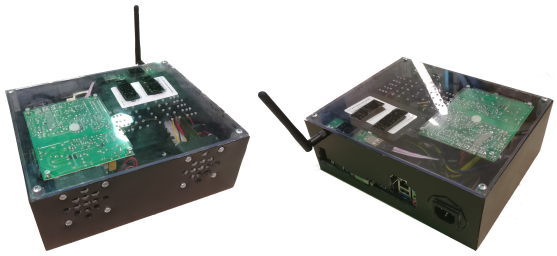
with

$$k = \frac{-xA - yB - zC - \frac{D}{C}C}{A^2 + B^2 + C^2}$$

After this correction, the point R is converted back to pixel coordinates that can be converted into inertial frame coordinates with homography matrices used by the 2D version of the localization system.

4.3 Remote Cameras

As mentioned before, the optical sensors used, either an RGB camera or a Kinect 360, can be connected to remote processing units. These assemblies, for simplicity, are called remote cameras. They are based on Micro-ATX boards with integrated x86 CPU's. A minimal installation of the Debian operating system, version 8, is running on each remote camera customized to integrate seamlessly with the ADIS system. They boot automatically when plugged to a power socket, connect to the ADIS Wi-Fi network during boot and run a custom camera controller software with elevated privileges.



(a) Remote Camera front view (b) Remote Camera back view

Figure 10: Remote Camera

The camera control software is written in Python and leverages the "OpenCV (Open Source Computer Vision Library)" compiled with "Intel TBB (Threading Building Blocks)" support and its python bindings for all image processing related tasks. To acquire image data from the cameras it uses the standard drives provided by the Linux kernel, whereas to acquire image and depth data from the Kinect 360, a customized driver based on the libfreenect[6] from the OpenKinect[7] community was used. The camera control software provides a text-based interface that can be view using a monitor plugged into the remote camera or remotely using an SSH connection. In addition to the text-based interface, the remote camera also starts its own web-server that serves a simple HTML web-page were a real-time video feed of the acquired data is visible using any browser. This video feed is clickable and switches between the an RGB video feed captured, the image processing results and, if a Kinect is connected, a colored depth map view.

4.4 Markers

The localization system relies on color-segmentation to detect the markers. Natively, the digital cameras and the Kinect provide images encoded in the RGB (Red, Green, Blue) color space. When the processing power is limited, for example, while running the localization software without remote cameras, the color-segmentation is done in the RGB color space too. But when using remote cameras the RGB color space is converted to HSV (Hue, Saturation, Value). The HSV color space provides better results with color segmentation using thresholds. RGB as to use all three channels to represent color, HSV represents

color only in one channel independent of brightness or saturation.

The markers are circles, five centimeters wide, made of colored paperboard. Their color was defined by analyzing the laboratory ambient lighting. From a picture taken with the ADIS system cameras, the least common color detected was pink with a hue around 300°.

4.5 Overlap Zones

The ADIS system is capable of using multiple cameras to cover a larger area. To do this without blind spots, either the cameras are perfectly aligned with each other or there are some areas that are covered by more than one camera. Inside these overlap zones, one marker is detected by multiple cameras, meaning that the system detects more points than it should. This would not be a problem if the localization system was perfect, and one marker spawned multiple detections with exactly the same coordinates, making it easy to correct the overlap zones just by coordinate comparison. However, the system has inherent errors that make different cameras detect the same marker in slightly different positions.

To solve this problem, the localization system analyzes the shape "drawn" by the markers instead of its location. Assuming that this perceived shape does not completely change with the camera point of view, it is possible to associate multiple detected points by analyzing the shape that they form. Procrustes analysis[8, 9] was used to interpret the aforementioned shape. This method gives a linear transformation (translation, reflection, rotation, and scaling) that best conforms a given set of points into another.

First, the overlap zones are identified by analyzing the field of view of each camera in the system. Then if during runtime any point is detected inside these overlap zones, the Procrustes analysis method is used to match all multiple detections of the same point. Finally, the extra points are eliminated, using one camera as reference, replacing the remaining correct points by a mean point calculated between all repetitions of the detected marker. Using a mean point smooths the transition inside and out of an overlap zone, aiding the marker tracking process.

4.6 Tracking

To track multiple markers while the ADIS system is running, a Kalman filter [10] based tracker was implemented. The tracker sole purpose is to associate all detected markers to distinct identities and follow these markers through time matching their identities correctly.

Using a simplistic view, a Kalman filter is an iterative algorithm that uses a series of measurements observed over time to estimate future measurements. These future measurements are used to further correct the Kalman filter, in order to improve its accuracy for the next prediction.

This behavior is used to build a tracker by associating each marker to its independent Kalman filter. Each filter predicts the next location of its marker. When new location data arrives, it is compared to the Kalman filters data and each detected maker is matched to its closest filter prediction (within a certain Euclidian distance). When all detected markers are matched, its location data is used to correct its corresponding filter. This process is repeated through time for all matched filters. When a filter is not matched with any marker a predefined number of times, this it's deleted, and its corresponding marker is considered as lost. The opposite also happens, when a new marker appears, a new Kalman filter is created to follow this marker until it disappears again, or the localization system stops tracking.

4.7 Calibration

Every time the ADIS system cameras change positions, a calibration process needs to be executed. This calibration process estimates the homography matrices used by the localization system to convert pixels to inertial frame coordinates. It is a semi-automated process, that relies on the placement of a calibration pattern in front of each camera. This calibration pattern feeds to the system a set o points, with known coordinates in the inertial frame. These points are detected by each camera and used to estimate each homography matrix using the singular value decomposition technique. An analysis was done to find the best location for the calibration pattern. Two sets of calibration points were defined, one set was scattered across the camera's entire field of view, and another was placed only at the center.

Pattern Position	Error	
	mean (cm)	standard deviation (cm)
Centered	4,1	4,7
Scattered	5,9	7,5

Table 1: Calibration Pattern Location Tests Results

The mean error, when the calibration pattern is centered, is around 2 centimeters bigger than when

the calibration points are scattered. The same is true for the standard deviation. This increase is most likely related to lens distortions around the edges. The SVD estimation fits lens distortions, increasing the overall error in the final estimation. These differences are not that large, comparing to the size of the vehicles in use, but they are big enough to choose a centered calibration pattern instead.

5 Vehicles

Although the ADIS system supports two types of vehicles, the main focus of this work revolved around the integration of flying vehicles, more precisely a quadcopter. The selected quadcopter, or drone, was an AR.Drone 2.0 from Parrot.

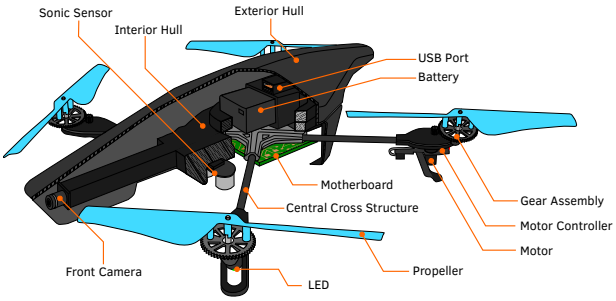


Figure 11: AR.Drone 2.0 cut-out

This drone comes equipped with an ARM processor, Video Digital Signal processor, 1GB of RAM, a Wi-Fi chip compatible with b/g/n Wi-Fi networks, a forward facing HD camera, a QVGA vertical camera, a pressure sensor, a USB 2.0 port, a 3 axis gyroscope, a 3 axis accelerometer, a 3 axis magnetometer, and an ultrasonic altimeter.

The internal operation of this drone was thoroughly analyzed and adapted to integrate the ADIS system. Using a *chroot jail*, an external Linux file system was connected to the USB 2.0 port. This file system runs a custom startup sequence that connects the drone to the ADIS Wi-Fi network with a defined IP address, and prepares it to be controlled by the user, using the drone's SDK[11] , or the Matlab's Simulink with the AR Drone Simulink Development-Kit V1.1[12] or even a custom binary package. This file system is also remotely accessible using a telnet client.

6 Communications

To send information from one computer to another the system relies upon two different protocols named TCP and UDP. Broadly speaking, the TCP protocol is used to send most of the messages exchanged within the ADIS system due to its reliability features. UDP is used only when communication speed is mandatory, more specifically, when location data is sent from the main computer to the navigation control algorithms and when the remote cameras are transmitting its location data to the localization software. The messages between the remote cameras and the localization software are encoded using a custom format fully documented.

The ADIS system is also capable of detecting all of its components connected to its network automatically. The detect its components, the ADIS system scans its entire network relying on the Address Resolution Protocol, ARP, defined by the RFC 826[13]. The scan mechanism sends a *ping* command across a predetermined interval of IP addresses, filling up a local ARP cache with detected IP addresses and the respective MAC addresses. These detected MAC addresses are then compared against a file where the MAC addresses of the ADIS system components are registered. This comparison identifies the available components on the network. These components can be vehicles or remote cameras.

The ADIS network has certain IP addresses reserved for certain components. This was done to facilitate the network scanning but also to avoid IP address conflicts between multiple network connected components.

7 Performance

In this section, a comprehensive set of tests is described and its results presented, with the intent of evaluating the overall system performance. Before starting the performance analysis a baseline for success was set. Every image capture device, either the USB camera or the Kinect works with a maximum frame rate of 30 frames per second. 30 Hz is also the speed at which the drone processes commands. So ideally the ADIS system should also work at a frequency of 30 Hz. A minimum sample rate of 20 Hz was set. Below this limit, the system is considered unfit to control a drone or a grounded vehicle. The drones dimensions are approximately 38x29x12 centimeters, and from all of the available vehicles, the smallest dimension is the drone's height. So empirically was set a maximum absolute position error of

12 centimeters for the localization system.

The test setup consisted of one forklift with a remote camera mounted on the tip of its fork. The fork was raised until the remote camera was 4 meters high. A tripod holding a 5 centimeters diameter marker was placed below the camera at a fixed height. A grid of 40x40cm squares was drawn on the ground occupying the entire field of view of the camera. The intersection between the grid lines defined the fixed measuring points to be used during the tests. For each point, 30 consecutive measures were taken, corresponding approximately to 1 second of acquisition. From these 30 measurements, the mode was assumed as the measured value. To analyze not just the system accuracy but also the data quality, the average ,and the standard deviation were also computed.

7.1 Digital Camera

When using a digital camera the localization system is only acquiring 2D location data, so just one test at 0 meters was performed. The results for this test,

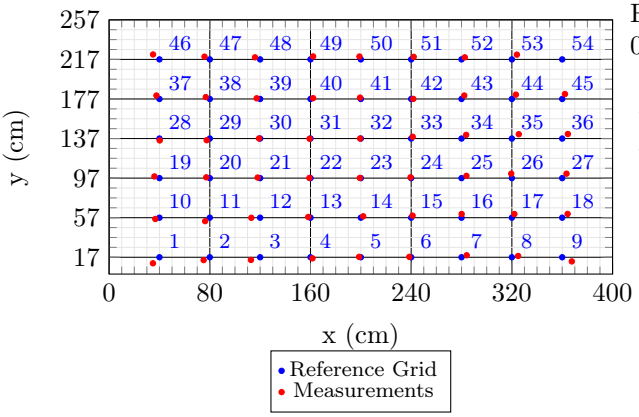


Figure 12: X and Y measured coordinates at 0 meters

are drawn in figures 12. Visually, all measured points are very close to their nominal values, and the closer the measurements get to the center of the image, the lower the accuracy error is. This is to be expected because no steps were done to correct lens distortions, unaccounted by the pin-hole model. These distortions are more pronounced in the edges of the lens and less pronounced close to the center. The accuracy error, was defined as the Euclidean Distance between the reference point, $p = (p_1, p_2, ..., p_n)$, and coordinates and its measurement $q = (q_1, q_2, ..., q_n)$:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Figure 13 plots the accuracy error for each measured point, and the standard deviation of the 30 samples gathered for each point measurement is plotted in figure 14 both for the x and y coordinates.

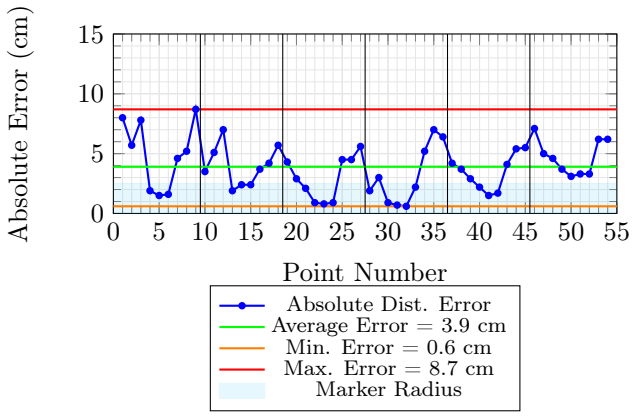


Figure 13: Absolute distance error for each point at 0 meters

differencent

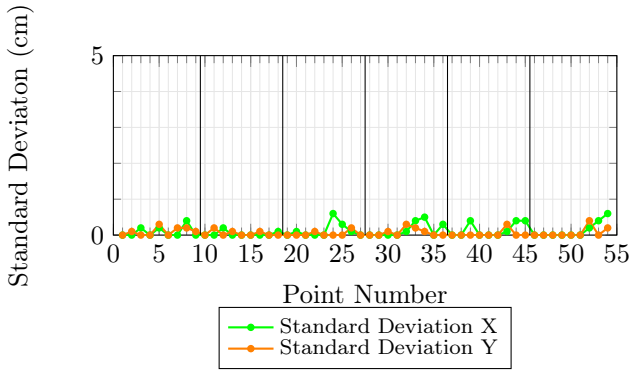


Figure 14: Standard Deviation for each coordinate at 0 meters

The maximum absolute error was 8.7 centimeters which is under the baseline value for success defined earlier of 12 centimeters. Furthermore, the average error is just 3.9 centimeters, meaning that the majority of the individual points were detected inside the marker. Figure 13 shows clearly the effects of lens distortion. In the center of the lens, the error dips considerable reaching errors below the 2.5 cm. A large

standard deviation reveals noisy measurements. Fortunately, the standard deviations measured are low, close to 0.5 centimeters, indicating low noise levels. Finally, it is important to note that the absolute error changes evenly along the image length, meaning that while a marker is being tracked, not abrupt changes in position occur that could affect the tracking algorithm performance or even change the relative positions between points across the captured area.

7.2 Kinect

The Kinect performance evaluation was done at four different heights, 0, 1, 1.5 and 2 meters. The measurements gathered and acquisition procedures were the same as used for the digital camera evaluation adding this time the height (z coordinate) measurement.

Because graphs similar figure 13 provide more information regarding the localization accuracy, graphs like figure 12 that show the accuracy error for each individual coordinate, will not be represented.

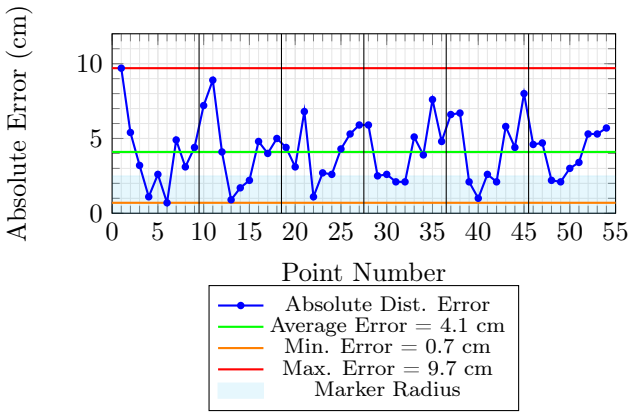


Figure 15: Absolute distance error for each point at 0 meters

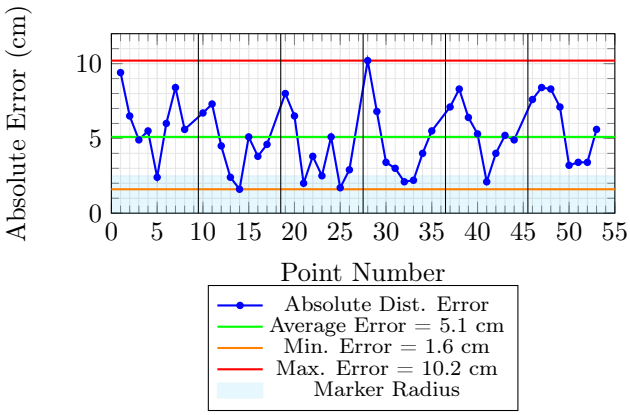


Figure 16: Absolute distance error for each point at 1 meter

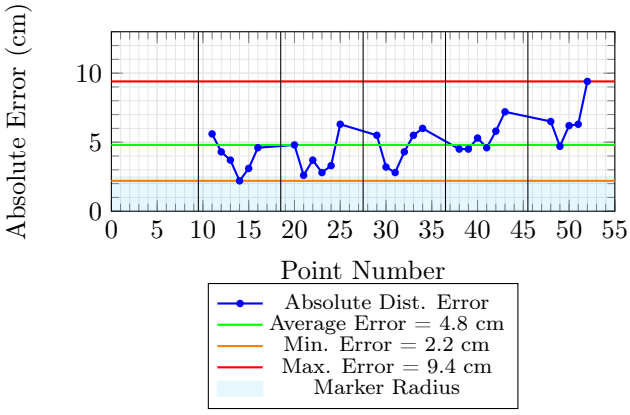


Figure 17: Absolute distance error for each point at 1.5 meters

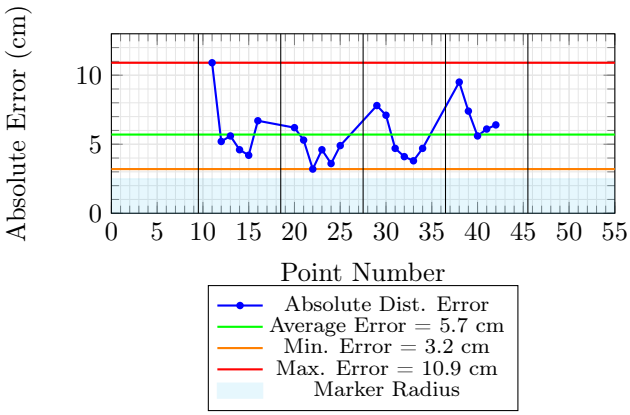


Figure 18: Absolute distance error for each point at 2 meters

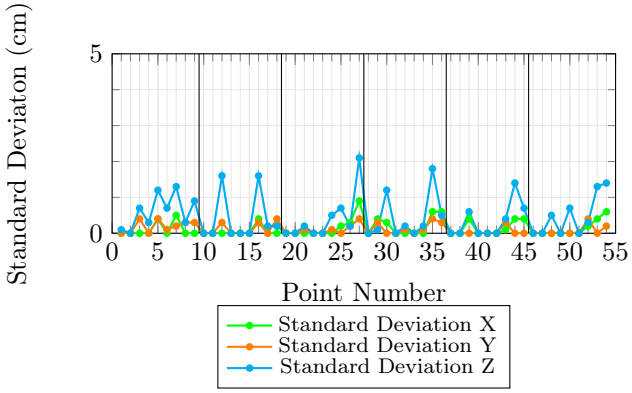


Figure 19: Standard Deviation for each coordinate at 0 meters

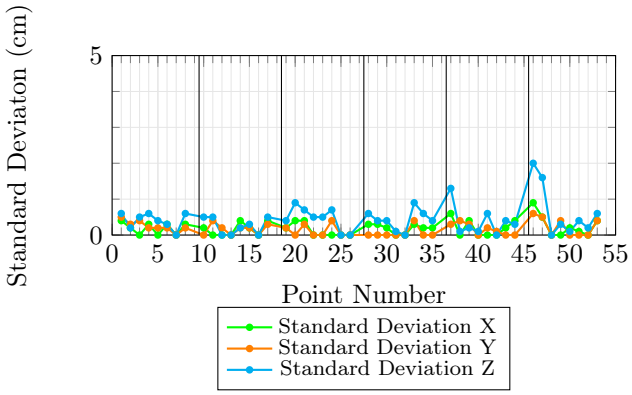


Figure 20: Standard Deviation for each coordinate at 1 meter

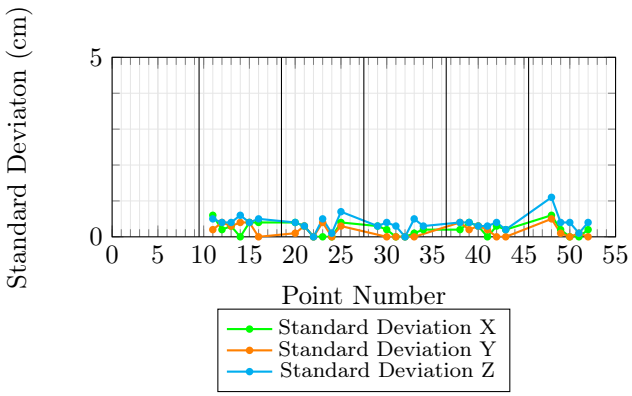


Figure 21: Standard Deviation for each coordinate at 1.5 meters

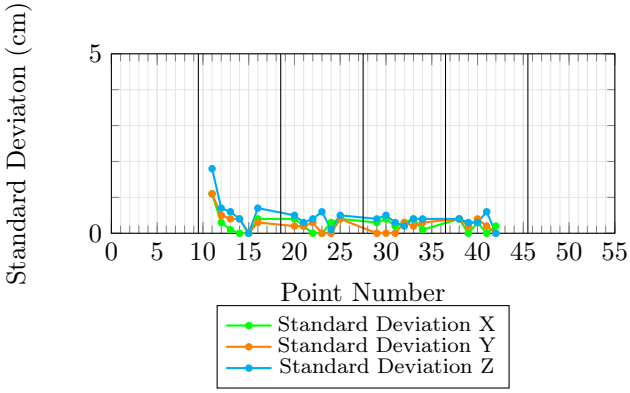


Figure 22: Standard Deviation for each coordinate at 2 meters

Across these four tests, the accuracy of the measurements stayed stable and consistent, showing that the ADIS systems behavior is kept through different height levels. The most dramatic difference is seen on the minimum error that increases from 0.7 centimeters to 3.2 centimeters. It is still just a 2.5 centimeters increase, that is small when comparing to the overall drone's size. However, it is enough to place the detected point outside of 2.5 centimeters radius of the maker, by 7 millimeters. Again these are small errors differences, but they need to be taken into account the user when using the ADIS system.

The average and the maximum errors change less from test to test. They also increase from the 0 meters test to the 2 meters test, but only by 1.6 centimeters and 1.2 centimeters respectively. The maximum error is consistently around 10 centimeters, the double of the average error which hovers around 5 centimeters. This error grows the higher the drone goes, is mainly caused by the x and y coordinates correction done by the system. When the marker is on to the ground, no coordinate correction is needed, so the system is effectively running similarly to the 2D mode using digital cameras. In fact, the results from the test at 0 meters with the Kinect and the test with the digital camera are extremely similar even though, they are different sensors. But when the marker is lifted from the floor, the calculations start to include the X and Y coordinate correction to account for the non zero height. This introduces errors, coming from imprecisions acquired during the calibration process, that increase the bigger the coordinates correction is.

The influence of the lens distortion is also visible since the less centered the points are, the higher the measured error is. This also worsens the results the higher the marker goes, because the area appre-

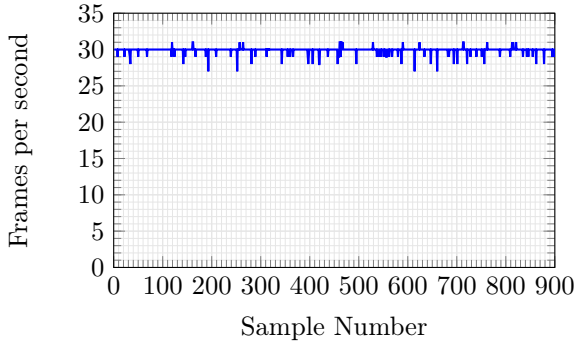


Figure 23: Frames per second test results

hended by the field of view is reduced. This makes the area below the center of the lens smaller, and consequently, less and less points can be inside this area where the lens distortion is minimal.

7.3 Sampling rate

To evaluate the localization system reliability and availability, frequency of the localization system's output was measured, with one Kinect connected to a remote camera with 5 markers in its field of view during 15 minutes. The output frequency was measured once per second by counting the number of packets received. Figure 23 shows the results.

The sampling rate rarely dropped below 30 frames per second and never reached any FPS number below 27. These last results are good and will provide robust location data to work with. Yet when the current version of the ADIS system was installed in the laboratory, occasionally results like the ones in figure 24 were found.

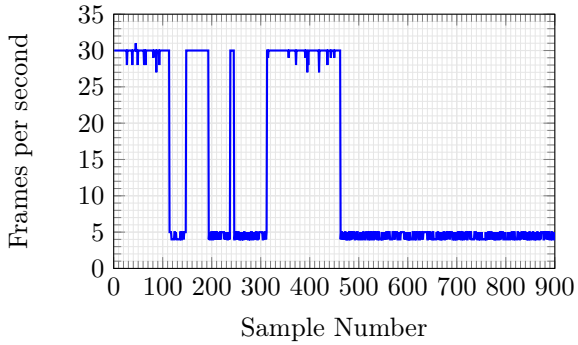


Figure 24: Frames per second test with bad connection results

After some debugging the root cause for this issue was pinpointed to poor router performance and high Wi-Fi network density, which were causing unexpected packet drops. This can be solved by replacing the current router with another, more powerful, model.

7.4 Tracking and Overlap Zones

Two tests were executed to evaluate how the tracking algorithm performs. One test consisted of using three cameras covering a large area and then drive a car with three markers on top to observe if any marker were lost or switched. This test evaluated the performance of the tracking algorithm in a 2D setup. The second test evaluated the tracking algorithm in a 3D setup. It used one remote camera equipped with a Kinect to track three markers on top of a drone. Both tests proved the reliability of the tracking algorithm, however, they also revealed three problems.

One happens when the car crossed over a thin area of an overlap zone, where one of the markers was suddenly labeled as new to the scene. This was caused due to the limits of the overlap zones not being as precise as they should. This problematic marker appeared as two, creating a new marker identification. because the car was already inside an overlap zone without the system's knowledge. Then when the car finally entered the overlap zone, the original marker was eliminated leaving the newly created one. The solution to this problem is to recalibrate the system, and if the problem persists, reposition the cameras to enlarge the overlap areas.

The second problem encountered is related to the Kinect depth performance. Usually, in dark-colored areas, the Kinect cannot measure depth information. This could lead to abrupt changes in the marker's location that lead the tracking algorithm to assume that a new marker appeared in the frame. This problem was solved by feeding into the tracking algorithm the x an y coordinates still not corrected for the z coordinate. This means that the tracking algorithm is the same used for 2D tracking and during testing it displayed the same performance.

The third problem is related to overlap zones with the Kinect. Unfortunately, these overlap zones cannot happen when using a Kinect. When two Kinects are pointing at the same area, their projected patterns interfere with each other. This interference prevents either Kinect to acquire depth information. This problem can be solved by synchronizing the depth capture process in such a way that multiple Kinects project their infrared pattern alternately. However, this would require a mechanism to rapidly

switch the infrared projector on and off, which is not currently supported by the custom Kinect driver used.

8 Conclusion

This thesis aimed to create a comprehensive system for the development of navigation and control algorithms targeted at mobile vehicles. The main idea was to provide a stable development environment, capable of accurate and reliable data collection for educational or research purposes. The vehicles used were both commercially available drones and radio controlled cars.

The system had to be able to locate and track more than one test vehicle in an indoor environment as well as communicate and control them. This led to the creation of an indoor localization system, a Wi-Fi based communication network and the customization of off-the-shelf drones to integrate the system.

The localization system can be configured to work in a 2D or 3D mode. After the tests presented in section 7, the localization system can be considered accurate enough to locate either a car or a flying drone. Nevertheless, problems still exist regarding overlap zones, because two Kinects interfere with each other when their field of view intercepts.

To follow the test vehicles through time, a tracking algorithm was implemented. Kalman filter based tracking. This tracking technique worked well in the right conditions. Unfortunately, edge cases caused by the overlap zones were also a problem. To handle these overlap zones, a specialized algorithm was integrated that uses the Procrustes method to find and delete duplicated points, however, this algorithm only starts when a point is detected as being within an overlap zone. Due to small inconsistencies between calibrated cameras, these overlap zones are calculated with small errors in their boundaries locations, which can originate duplicated detections and consequently matching errors.

To send and retrieve information from vehicles to fulfill any kind of communication with a remote device, a Wi-Fi based communication network was created. The main disadvantage of this approach was in situations with high network density caused by other networks. This impact occasionally caused large amounts of dropped UDP packets, slowing down the localization system.

Regarding vehicle integration, only flying vehicles were addressed. The drone model selected was the AR.Drone 2.0 from Parrot. This was a good choice because is durable, and it endured testing, and the official and unofficial community support helped the overall system integration considerably.

References

[1] M. Nuno. Integration of RC Vehicles in a Robotic Arena. Master Thesis, November 2016.

[2] W. Gander and J. Hrebicek. Solving Problems in Scientific Computing using Matlab and Maple.

[3] Xbox 360 release date "http://gizmodo.com/5563148/microsoft-xbox-360-kinect-launches-november-4"

[4] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," Communications of the ACM, vol. 24, no. 6, pp. 381–395, 1981.

[5] About OpenCV, "http://opencv.org/about.html"

[6] Libfreenect GitHub, "https://github.com/OpenKinect/libfreenect"

[7] OpenKinect community page, "https://openkinect.org"

[8] Kendall, David G. "A Survey of the Statistical Theory of Shape." Statistical Science. Vol. 4, No. 2, 1989, pp. 87–99

[9] Bookstein, Fred L. Morphometric Tools for Landmark Data. Cambridge, UK: Cambridge University Press, 1991

[10] R. E. Kalman, A New Approach to Linear Filtering and Prediction Problems,1960

[11] Stephane Piskorsk, Nicolas Brulez, Pierre Eline, Frederic D'Haeyer. *AR.Drone Developer* Guide. Version 2.0.

[12] AR Drone Simulink Development-Kit V1.1 ,"https://www.mathworks.com/matlabcentral/fileexchange/43719-ar-drone-simulink-development-kit-v1-1?requestedDomain=www.mathworks.com"

[13] David C. Plummer (November 1982). "RFC 826, An Ethernet Address Resolution Protocol -- or -- Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware". Internet Engineering Task Force, Network Working Group.