ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for
Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme
und Regelungstechnik

Lucas Liebenwein

Autonomous Pairing Of Distributed Flight Array Modules

Bachelor Thesis

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Max Kriegleder
Prof. Dr. Raffaello D'Andrea

September 2015

IDSC-RD-MK-16

Contents

Abstract

In this thesis an algorithm is proposed to let modules of a modular robotic platform, the Distributed Flight Array from ETH Zurich, pair up and then dock to their partner. During this process, modules first localize each other by estimating the bearing of their peers, then they will choose a partner among the modules within their neighborhood, approach this partner and dock to it with the use of magnets. For the localization task IR–transceivers are used to determine when two rotating modules are line–of–sight and a gyroscope then provides the current orientation angle. Modules are able to move on the ground with the help of omnidirectional wheels. To begin with, we introduce various hardware features and their corresponding application, namely driving on the ground, communication between modules, bearing estimation and collision detection to detect when two modules reach each other. Those elements play a key role in the pairing task. Subsequently, the method for two partners to dock to each other is presented followed by the algorithm that allows modules to choose a partner. It is based on the information a module gathers from the modules in its neighborhood. Different weaknesses and flaws where the algorithm does not find the maximum amount of possible pairs are evaluated as well. At the end, we analyze the results from experiments conducted around the pairing task.

Nomenclature

Symbols

a_i	node/module i	$[-]$
$\mathcal{A}$	availability graph	$[-]$
$\mathcal{A}_L$	set of modules with least available neighbors	$[-]$
$\mathcal{N}$	neighbor graph	$[-]$
$A(a_i)$	set of available neighbors of a_i	$[-]$
$\lvert A(a_i)\rvert$	number of available neighbors of a_i	$[-]$
$T(a_i)$	set of total neighbors of a_i	$[-]$
$\lvert T(a_i)\rvert$	number of total neighbors of a_i	$[-]$
$\nu(\mathcal{G})$	number of matches in a maximum matching	$[-]$
$\vec{a} = (\ddot{x}, \ddot{y}, \ddot{z})$	Acceleration of a module	$[\mathrm{m/s^2}]$
$\vec{a}_{xy} = (\ddot{x}, \ddot{y})$	Ground acceleration of module	$[\mathrm{m/s^2}]$
α	Orientation angle of module	[rad]
$\dot{\alpha}$	Angular velocity of module	$[\mathrm{rad/s}]$
d	Distance between center of module and wheels	[m]
I	Inertial frame	$[-]$
M	Body fixed frame	$[-]$
θ	Angle between body and wheel 1 of a module	[rad]
$\vec{v} = (\dot{x}, \dot{y}, \dot{\alpha})$	Velocity and angular velocity of a module	$[\mathrm{m/s}, \mathrm{m/s}, \mathrm{rad/s}]$
$\vec{x} = (x, y, \alpha)$	Position and orientation of a module	[m, m, rad]
(x, y)	Position of a module	[m]
$(\dot{x}, \dot{y})$	Translational velocity of a module	$[\mathrm{m/s}]$

Indicies

M	vector in body fixed frame
I	vector in inertial frame

Acronyms and Abbreviations

COBS	Consistent overhead byte stuffing
CRC	Cyclic redundancy check
DFA	Distributed Flight Array
ETH	Swiss Federal Institute of Technology
ID	Identification number
IMU	Inertial measurement unit
IR	Infrared
RANSAC	Random sample consensus
UML	Unified modelling language

Chapter 1

Introduction

In modular robotics a typical subject is for individual units to come together, assemble into a bigger structure and then perform a corporate task they are incapable of doing alone. One such modular robot is the Distributed Flight Array (DFA), a platform for research in distributed control and estimation developed at ETH Zurich. The DFA consists of multiple hexagonally shaped units, called modules, that can drive on the ground with omnidirectional wheels, communicate with other modules wirelessly (IR) or via a wired connection, dock with the help of magnets and then take coordinated flight using propellers, one of them attached to each unit. This platform is characterized by the fact that modules need to join together to be capable of flying since they are unable to individually fly.

This report aims to investigate the task of joining together, commonly called self–assembly, which might act as a starting point for any subsequent coordinated action. The DFA is already able to self–assemble making use of a modified version of a rendezvous algorithm [8] but only in random, non–specified configurations. The main objective here is to explore whether the DFA may also self–assemble in predefined structures.
For that we present a method to let two modules specifically dock to each other on a desired side. Afterwards, this approach is generalized so that any arbitrary number of modules can pair up and dock to their respective partner. The method presented comprises a set of features that were defined during the design process: The algorithm is scalable in a sense that regardless of the number of modules used it is always applicable, which allows to create any desired structure without limitation on the amount of modules included. It is distributed so that each module computes the necessary steps on its own with information it gathers from its neighborhood. This is done to keep the approach scalable but also to avoid any global communication. The algorithm was also designed to be homogeneous in a way that all modules abide by the same rules. This increases the flexibility since any module is then fully replaceable. Moreover, modules apply the actions in a parallelized manner where all modules act simultaneously instead of sequentially avoiding the otherwise necessary agreement on a docking sequence but also speeding up the process. Towards the self–assembly of DFA modules in desired configurations the results obtained from this work might serve as a first step in that direction.

Several modular platforms have already shown self–assembly in predefined shaped under various circumstances and with different kind of sensors [6]. Some of them use IR–sensors to communicate [15], [9], some rely on cameras [16], [5], [12] or radio communication [11]. Most systems depend on a seed to initiate the process where one unit after another is recruited by the seed and docks to it to achieve the final shape. In contrast, this report proposes a method where no seed is required and modules dock to each other in parallel rather than in a linear fashion.

We begin with reviewing the hardware of the DFA and especially the aspects relevant for self–assembly in Chapter 2. The respective purpose of the various features is explained as well. In

the following, Chapter 3 explains the approach to let modules pair up and dock to their partner. First, the state machine to let two partners dock to each other is presented, then the algorithm that enables multiple modules to find a partner is explained and strengths as well as weaknesses are highlighted. To analyze the proposed method experiments were conducted whose results are discussed in Chapter 4. Finally, the thesis is concluded in Chapter 5.

Chapter 2

Relevant Hardware Features And Their Corresponding Purpose

In this chapter the hardware of the DFA is presented. We give a general overview of the relevant hardware aspects, see Section 2.1, followed by a closer description of the various parts that are especially important for letting modules autonomously dock to each other and their respective application. The kinematic relations for driving and the corresponding controller that was developed is described in Section 2.2 and Section 2.3 discusses the various communication capabilities. Bearing estimation of neighboring modules is depicted in Section 2.4 and collision detection between modules is described in Section 2.5.

2.1 The DFA Hardware

As described in [13] the modules of the DFA are hexagonally shaped, symmetric, weigh 255 g and their side to side length is 250 mm. Three omnidirectional wheels, each driven by a 0.5 W brushed DC motor, provide holonomic 2D–motion. For docking two modules, four magnets are installed on each side to hold modules together once they are sufficiently aligned. Moreover, a number of sensors and communication devices is included for various purposes and in the following the ones relevant for the self–assembly task are listed:

- Inertial measurement unit (IMU) including a three–axis digital MEMS rate gyroscope and accelerometer.
- Line of sight IR–transceivers on each of the six sides, called ports, with a range of about 1m and a transmission cone of roughly 5° used for communication purposes.
- Spring–loaded pins for communication via a UART–interface when modules are docked with each other.
- Low–latency, unidirectional broadcast system that is used to transmit user–commands from a base station.

The inertial measurement unit, the communication devices and the driving capabilities are of particular interest in this context since they play a significant role during the pairing task. Their purpose (sensing, communicating and driving) is discussed in the remaining sections of this chapter. All aspects described above are completely identical on every module allowing the pairing algorithm, which will be introduced later, to be homogeneous as desired. Figure 2.1 (A) provides an overview of a module's hardware.

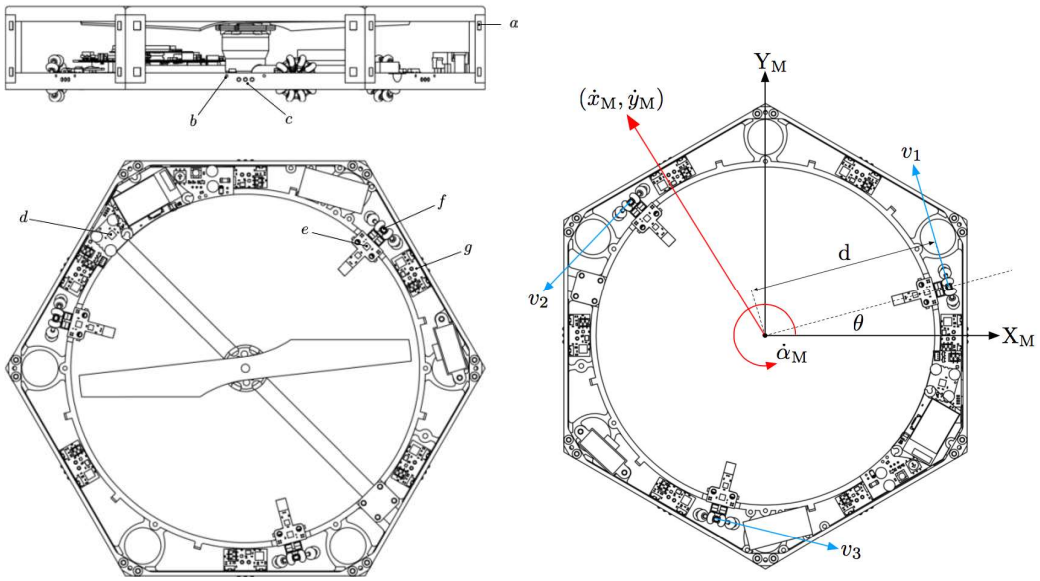


Figure 2.1: The hardware of a module and the coordinate frame to navigate a module is shown. (A) The following parts are necessary for two modules to assemble: (a) Magnets to dock to each other, (b) IR–transceivers to communicate with modules within the sensing range, (c) pins to communicate in the docked state, (d) the inertial measurement unit, which houses the gyroscope and the accelerometer, (e) H–bridge electronics, (f) omnidirectional wheels for holonomic motion, (g) electronics for communication via pins and IR [13].
(B) The figure shows how the body–fixed coordinate system M is orientated. The omnidirectional wheels, each separated to the others by 120°, are located at a distance d away from the center and the first wheel is rotated at an angle $\theta = 15°$ w.r.t. X_M. The body frame is rotated w.r.t. to the inertial frame (not shown here) by the angle α in positive z–direction [8].

2.2 Kinematics

For driving on the ground a module possesses three omnidirectional wheels, each separated by 120°. Those wheels allow a module to translate and rotate in any desired direction, even at the same time. However, the pairing task only requires modules to either rotate or translate.

2.2.1 Basic Relations

Velocities are described in a body–fixed coordinate system, denoted by M, that is located in the geometric center of the module, see Figure 2.1 (B). A module's velocity $\vec{v} = (\dot{x}, \dot{y}, \dot{\alpha})$ consists of two components for the translational motion $(\dot{x}, \dot{y})$ and one component for the rotation $(\dot{\alpha})$. The relation between $\vec{v}_M$ in the body frame and the individual wheel velocities (v_1, v_2, v_3) is as follows [8]:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \mathbf{T}(\theta) \cdot \vec{v} = \begin{bmatrix} -sin(\theta) & cos(\theta) & d \\ -sin(\frac{\pi}{3} - \theta) & -cos(\frac{\pi}{3} - \theta) & d \\ sin(\frac{2\pi}{3} - \theta) & cos(\frac{2\pi}{3} - \theta) & d \end{bmatrix} \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \end{bmatrix} \tag{2.1}$$

The physical parameter d describes the distance between the origin of the coordinate frame M and each wheel, whereas θ is the angle between the X–axis of the coordinate frame M and the wheel closest to it, see Figure 2.1 (B).

To navigate a module an inertial frame I is introduced as well, which is chosen such that the initial body frame corresponds to the inertial frame. The inertial frame is used to navigate modules and the generated velocities are transformed accordingly to wheel velocities via the respective transformations. Since the module is assumed to operate on a perfectly flat surface any rotation between the inertial and the body frame can be neglected except for the rotation around the common z–axis that is perpendicular to the surface. To transform a vector from the inertial frame to the body frame the following transformation, i.e. a rotation about the inertial z–axis, is used:

$$\vec{v}_M = \begin{bmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \vec{v}_I, \tag{2.2}$$

where α denotes the angle about the inertial z–axis. Using the transformations (2.1) and (2.2) the generated velocities expressed in the inertial frame can be transformed to the according wheel velocities.

2.2.2 Estimation Of The Current Orientation Angle

To determine α, i.e. the relative angle from the initial orientation, the angular velocity $\dot{\alpha}$, which can be measured directly with the gyroscope, is numerically integrated to estimate the current orientation α of the module. Since such IMUs are susceptible to drift, the gyroscope is constantly recalibrated when the module is not moving. This can be determined by the current value of the velocity input. Also, to account for the fact that a module might be moved manually, the accelerometer is checked for whether any motion can be detected. When the norm $|\vec{a}|$ of the acceleration $\vec{a} = (\ddot{x}, \ddot{y}, \ddot{z})$ exceeds a certain threshold, the gyroscope will not be recalibrated until the module is not moved anymore. The accelerometer, on the other hand, was calibrated once off board and is not recalibrated on board.

2.2.3 Drive Controller

In order to navigate a module, it is desirable to have feedback on the three degrees of freedom in 2D–space, denoted here by x, y and α or in vectorial form $\vec{x} = (x, y, \alpha)$. This would allow one to use a control loop on those three states to accurately approach a desired position. In fact however, there is no feedback on the position of a module or the translational velocities available onboard. Thus any desired translational motion is simply fed forward to the wheels and there is no information about the actual distance a module has moved. On the opposite though, the current orientation α of the module is estimated with the data acquired from the gyroscope as described above. This information is used in a manually tuned PID control loop to achieve the desired angle when rotating. During translation a PD–controller, also manually tuned, keeps the orientation of the module constant. The I–Part was found to be redundant in this case. Having control over the orientation of a module is essential for the docking process described in the next chapter (Chapter 3) to work properly.

2.2.4 Experimental Analysis

To analyze the performance of the controller for orientation we conducted various experiments where a single module moved on the ground. The experiments took place in the Flying Machine Arena of ETH Zurich that features a high–precision tracking system [10], which was used to gain ground truth data of the current position and orientation of the module.
Five identical experiments were conducted in which a module was supposed to rotate for 360°, see Table 2.1 for the results and Figure 2.2 for the error of the onboard orientation estimation. To evaluate the error of the onboard orientation estimation the onboard data was synchronized manually with the ground truth data during post–processing. In all experiments the module rotated too far with the error varying approximately between 2° and 4°. It can also be seen that the orientation angle is underestimated by about 2° to 5° after rotating but never overestimated

and that it oscillates during the rotation phase. This may explain why the module rotates too far. Moreover, it is worth noting that in three out of the five experiments the error of the estimation is almost identical and in a fourth experiment the same oscillation may also be seen but shifted by 2°. This suggests that there is a systematical error present but to fully understand the behavior of the error further analysis would be required.
A second set of five experiments is shown in Figure 2.3 where a module was supposed to move 1 m without rotation in a desired direction. After the module starts moving, it is rotated by about 1° in all experiments but afterwards the orientation remains constant. Since there is no integral feedback action for the angle during translation the error is not compensated for. Furthermore, the error of the onboard orientation estimation is significantly below the error during rotation. This might be due to less drift in the gyroscope during translation. As stated before in this section there is no feedback on translation. When looking at the position of the module it can be seen that it does not move far enough. One possible explanation could be that the wheels are modeled without slipping but slipping is actually present. In the direction normal to the motion the module drifts up to approximately 2 $^{\text{cm}}/_{\text{m}}$.
The entire data has been lowpass–filtered during post–processing to reduce the effect of noise in the figures. Additional experiments to analyze the performance of the drive controller have been conducted and can be seen in the Appendix A.

	Final orientation angle [°]	Error to desired orientation angle [°]
1	363.59	3.59
2	361.98	1.98
3	362.25	2.25
4	361.69	1.69
5	362.05	2.05

Table 2.1: The final orientation angle and the error to the desired orientation angle for the conducted experiments is shown (ground truth data). In all experiments the module was supposed to rotate 360° but goes slightly further as can be seen.

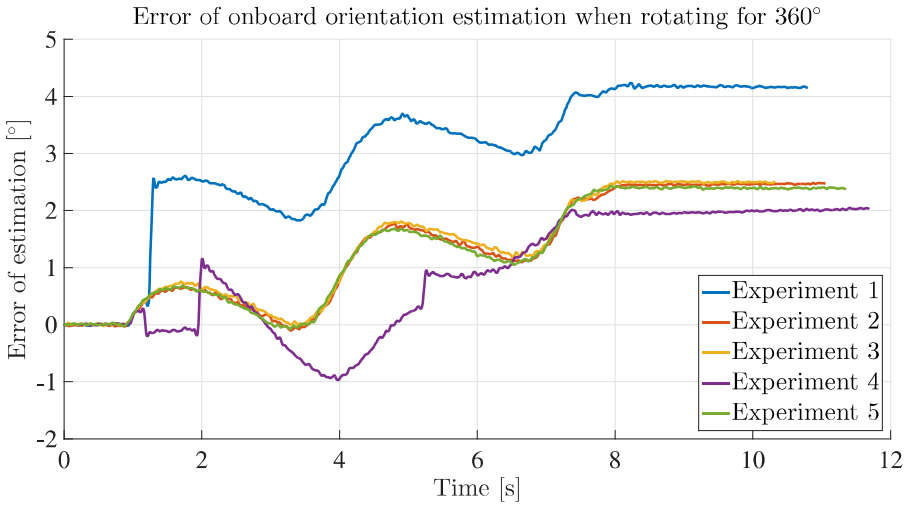


Figure 2.2: The onboard estimation is compared to the actual orientation and the error is evaluated. In each experiment the module is supposed to rotate 360°. A positive error corresponds to an underestimated angle.

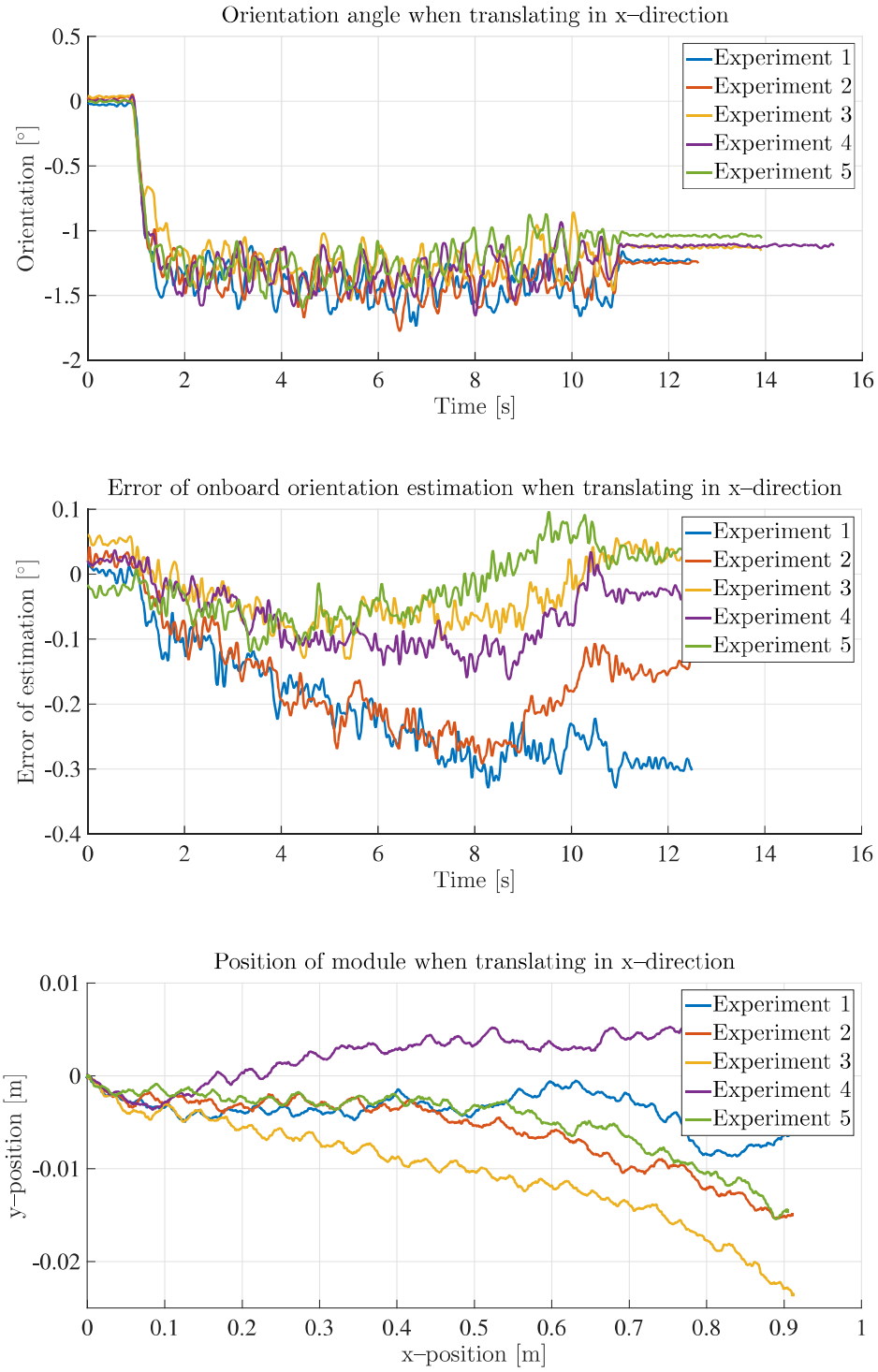


Figure 2.3: In all five experiments shown above the module was supposed to move 1 m in x–direction (body frame) without rotating. The initial position and orientation was taken to be 0. In the top plot the orientation of the module during the experiment is shown. The break in the curve corresponds to the moment when the module starts moving. In the middle the error of the onboard orientation estimation is shown and the lower plot displays the position of the module.

2.3 Communication

Modules can communicate with each other by using their built–in communication capabilities. When modules are not docked they use the IR–communication to exchange messages. Those line–of–sight transceivers transmit in a cone of about 5° and have a range of approximately 1 m. Modules are considered to be neighbors when they are able to sense each other via IR. When modules rotate on the spot while transmitting it becomes more likely that two modules are in sight of each other, at least temporarily, since the IR–transceivers do not transmit in all directions. Therefore, whenever messages are transmitted via IR, modules will also rotate at the same time. Once modules are docked they can communicate via the pins on the according port. The communication via pins is also used to confirm the docking between two modules. Messages will be constantly sent out via the pins and when a message is received back the docking is considered successful. Any message sent out will be bundled in a consistent overhead byte stuffing (COBS) packet and a cyclic redundancy check (CRC) is added. With COBS the receiving module can identify single packets within a data stream by a unique starting sequence [1]. Any packet that is received can be checked for completeness as well as correct transmission via a CRC [7].

2.4 Bearing Estimation

2.4.1 Working Principle

Another crucial feature for letting two modules dock to each other is their ability to sense the bearing of neighboring modules. This is done by applying the following steps:

1. Modules rotate on the spot and periodically transmit messages via all IR–transceivers. Those messages will at least contain the unique ID of the transmitting module.

2. Simultaneously each module receives IR–messages from the other modules. Those messages are captured together with the current orientation angle yielding an estimation for the direction the packet came from and therefore the bearing of the neighboring module.

3. Afterwards, received messages are evaluated separately for each port where a packet was received. From each port the maximum and minimum value for the orientation angle are taken and those two are then averaged. Left with up to six bearing estimations (there are six ports), those values are run through a random sample consensus (RANSAC) algorithm to detect outliers, neglect them and to average the remaining ones [4]. This yields an estimation for the bearing of neighboring modules.

Since each module possesses an unique ID, we can determine the bearing of several modules at the same time by applying above steps for each neighboring module individually.

2.4.2 Experimental Analysis

Similarly to the previous section, experiments were conducted in the Flying Machine Arena to acquire ground truth data of position and orientation of modules [10]. In these experiments, two modules were randomly placed apart from each other but within the sensing range and both modules sensed the bearing of the other. We then compared the results to the ground truth data. It was also evaluated how the RANSAC algorithm improves the result compared to a simple average of each port measurement.
Ten experiments were conducted, see Figure 2.4, and the bearing estimation of both modules was evaluated (results 1-10 from module one and results 11-20 from module two) with the error of the overall estimation varying between approximately 0° and 25°. The overall estimation, as stated before, results from the average of the estimations from each port, which are taken as the average of the minimum and maximum value acquired from the respective port. It can be seen that in most experiments there are less than six single estimations although there are six ports, which was found to be due to the unreliable IR–communication. One reason for the unreliable IR–communication

is that since the IR–transceivers are line–of–sight the region to receive the signal is rather small. This makes it more unlikely for modules to always exactly meet in a configuration during rotation (regarding orientation towards each other) where they can sense one another. Also, the transceivers cannot transmit as well as receive at the same time and the communication between modules is not synchronized. Therefore, it can happen for modules to transmit and receive at the same time but then no message at all comes through.
Concerning the RANSAC algorithm no improvement over the simple average is identified for most cases. Typically, the single estimations from each port were spread over an interval of about 10°. For instances with significant outliers on the other hand, as in experiment 18 in Figure 2.4, the outlier is detected and removed from the average.
The spread of the estimations between ports but also the actual error can be partially explained due to the type of the IR–transceivers (line–of–sight, cone–like shaped transmission region) and the geometry of the modules (hexagon). In the ideal situation the data gathered from one port is spread over a certain interval but remains centered around the actual bearing. When two modules are initially orientated randomly and then start rotating at the same speed, the following situation may occur as well: Modules only sense each other in a partial interval of the ideal interval causing the data to be centered around a different value than the actual bearing. Another source of error is certainly the drift in the estimation of the orientation angle.

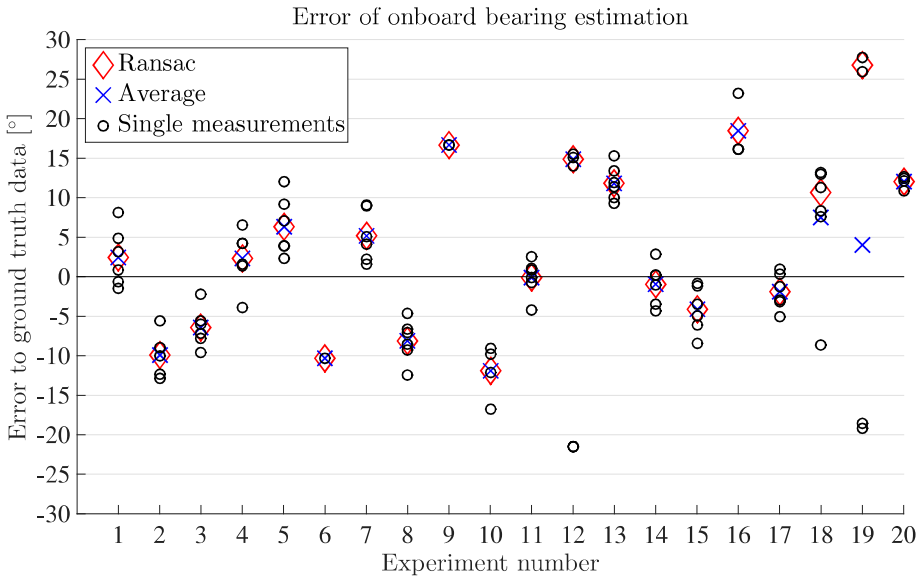


Figure 2.4: The onboard bearing estimation is compared to ground truth data. Two modules were used to sense the bearing of each other (experiment number 1–10 for one module and 11–20 for the other module). All data is plotted w.r.t the error to the ground truth data. The single estimations from each port are shown as well as the RANSAC result and the average of all estimations.

2.5 Collision Detection

When docking to each other as described in Chapter 3, two modules must be able to determine when they have reached each other. Since there is no information about the distance they are apart from each other nor is there any reliable feedback on the distance a module is displaced, they instead actively monitor collisions, which can be achieved with the onboard accelerometer.

2.5.1 Working Principle

When one module is moving towards another one and they hit each other, the collision in this moment can be sensed. In particular, we consider the Euclidean norm of the acceleration in x– and y–direction, i.e. $|\vec{a}_{xy}| = |(\ddot{x}, \ddot{y})|$, where x and y are the two directions of translational motion, see Figure 2.1 (B). The following steps are applied to check for a collision:

1. A module keeps the latest set of $|\vec{a}_{xy}|$ samples with $\ddot{x}$ and $\ddot{y}$ acquired from the accelerometer.
2. If within this set at least three of the samples are above a certain threshold, the event is considered a collision. The actual threshold depends on whether the module is moving or not.

Due to the eccentric placement of the IMU collisions may only be detected reliably when the module does not rotate but only translates. This is because during rotation the IMU is subject to centripetal forces that increase quadratically with increasing angular velocity ($\sim \dot{\alpha}^2$) making collision detection by a single threshold more difficult. Further, the collision detection is only to be used when the module is uniformly moving since the accelerations achieved during accelerating a module to the desired speed are of similar magnitude as the chosen threshold. To account for that, the detection is only activated shortly after the motion has reached a steady state. It must be noted that modules cannot distinguish between different kinds of collisions. Therefore, any collision is treated the same way and the module will assume it just hit another module, regardless whether it actually hit a module or just any random object, e.g. an obstacle.

2.5.2 Experimental Analysis

The collision detection mechanism was developed based on the information gathered from experiments in which modules drove against each other to simulate a collision. We then evaluated the norm of the acceleration in x– and y–direction. One set was conducted for a module approaching another module and a second one for a module expecting to be approached, see Figure 2.5. It can be seen that the peak acceleration highly varies between experiments, which we assume to be out of two main reasons: First, the peak depends on the orientation of modules and the eccentric placement of the accelerometer causing different damping in each situation. Second, it might not be guaranteed that the actual peak of the collision is sampled.
Since three samples must be above the chosen threshold to consider the event a collision, not only the peak but also the third largest value during the collision, called threshold value, is examined. It is found that even the lowest threshold value is clearly apart from the average noise. Therefore, it was chosen to set the threshold about 20% below the lowest threshold value.
Additional experiments regarding the collision detection were conducted to further analyze the reliability of the mechanism, see Appendix A.

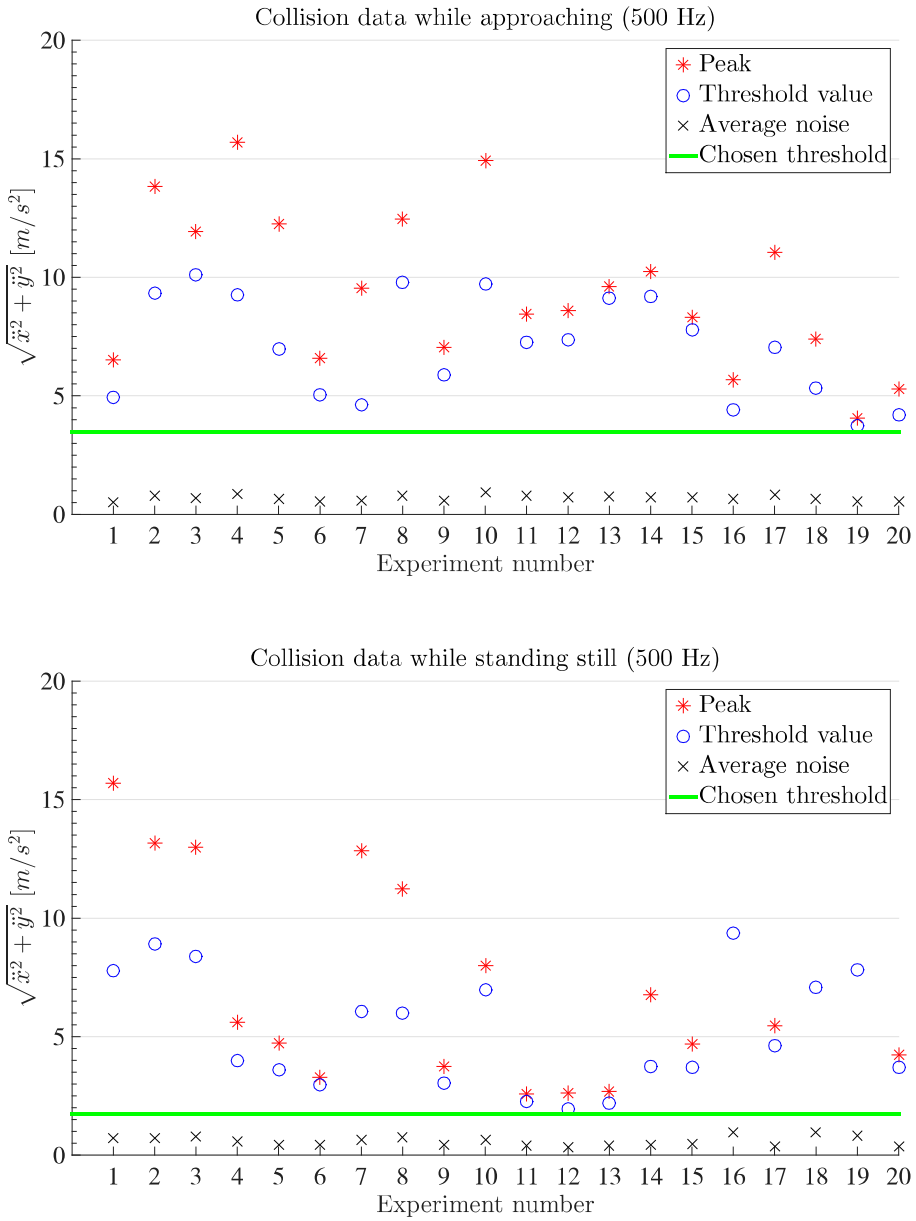


Figure 2.5: For detecting collisions the norm of the acceleration in x– and y–direction is analyzed. For each experiment the peak acceleration is shown as well as the third highest value (threshold value) and the average noise. It is distinguished between an approaching module (upper plot) and a module that is standing still when the collision occurs (lower plot).

Chapter 3

Pairing Algorithm

The following chapter presents the algorithm that is used to let modules autonomously pair up and dock to each other. In Section 3.1 we explain the approach developed for two modules to dock to each other based on the corresponding state machine. This work is built upon the results from the previous chapter where the key features (sensing, communicating and driving) for self–assembly were closely examined. Section 3.2 discusses how the approach for two modules can be generalized to an arbitrary number of modules. The goal is that every module finds a partner and docks to its partner afterwards, see Figure 3.1.

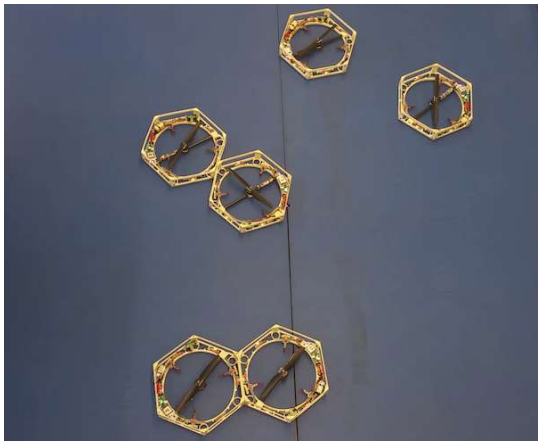


Figure 3.1: Six modules are currently in the process of docking to their partner. One pair has already successfully docked and one is about to dock, whereas the third pair is still separated. Those two are currently approaching each other and will eventually also dock. Once every module has paired up, the process is completed.

3.1 Autonomous Docking Of Two Modules

Towards the goal to have multiple modules pair up, we first evaluate how two modules can stably dock to each other. This can later be used to let any number of modules dock to their corresponding partner once they have found one. A number of specifications for this task is addressed in Section 3.1.1 and the state machine is presented afterwards in Section 3.1.2. The algorithm was developed with the help of continuous testing where the state machine was implemented on several DFA modules but not with simulations.

3.1.1 Prerequisites

As mentioned in Chapter 1 the approach is supposed to be distributed, homogeneous on all modules, scalable and parallelized. For the task of docking two modules some of the specifications drop out since the number of modules is limited to two. Therefore, especially the conditions for having a distributed and homogeneous algorithm are of importance. Throughout the process the abilities to communicate, sense the bearing and detect collisions are thoroughly used. In Figure 3.2 a possible configuration in the initial and final state of two modules docking to each other is shown.

(a) (b)

Figure 3.2: Two modules are shown before (a) and after (b) they have docked to each other. The initial configuration can be arbitrary but modules must be able to sense each other with the use of the IR–transceivers.

3.1.2 State Machine

The state machine, based on a UML approach [14], consists of seven different states, which are sufficient to let two modules dock, see Figure 3.3. A short overview of all states will be given followed by a closer description of each of them. At the beginning, the two modules will try to sense the bearing of each other. Afterwards, they are going to orientate towards each other and then one of them is going to approach the other one. When they have reached each other, they will try to dock. If the docking is successful the process is finished, otherwise they will back up and repeat the procedure. Also in case they have not reached each other during the approach they will repeat the procedure.

Idle state

In the beginning, modules will be in the idle state where they do not move or do anything else except for awaiting an external signal from the user to initiate the process. This signal is transmitted via the unidirectional broadcast system described in Section 2.1. This triggers the transition to the next state, the *find* state.

Find state

Both modules will rotate on the spot and try to sense the other one by receiving messages from the other module via the IR–transceivers. This will look somewhat similar to what can be seen in Figure 3.2 (a). At the same time modules estimate the bearing following the approach described in Section 2.4, where the combined input from the gyroscope and the IR–communication is used. It was found to be sufficient to let modules rotate for 360° so that packets from the other module should be received on each port at least within one time interval. In case modules cannot sense each other, e.g. because they are not within range of each other, they will go back to be idle. Otherwise they are going to orientate towards each other. To keep the action of both modules synchronized the respective transition is only allowed once a timeout is triggered. The timeout is

based on the expected duration of the state. It is to be noted that all subsequent transitions work in a similar matter for the same reason.

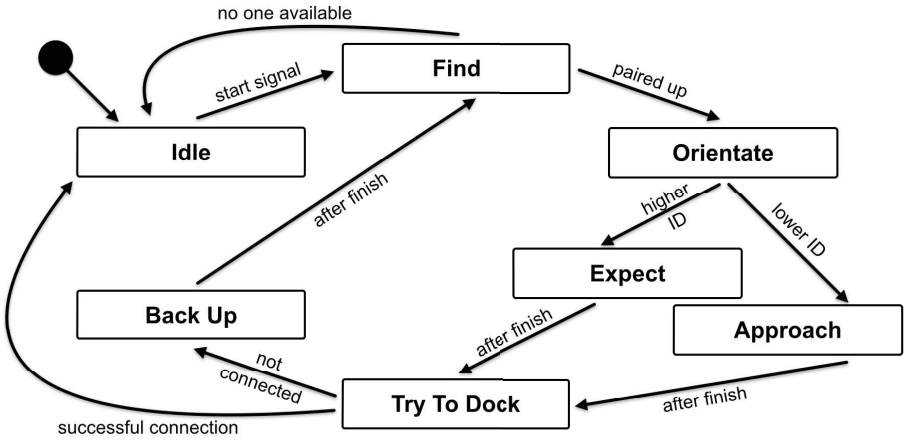
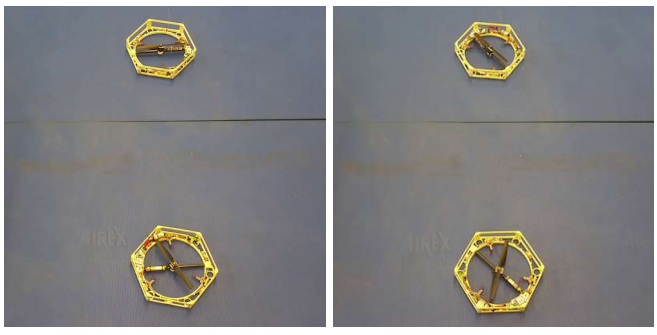


Figure 3.3: This figure shows a graphical representation of the state machine for letting two modules autonomously dock to each other. Rounded rectangles correspond to states, arrows in between states indicate transitions and the according label marks the condition necessary for the transition to take place. The initial state (idle state) is marked by a filled circle.

Orientate state

For each docking module we can specify one of the six ports to be used for docking. In this state both will orientate the specified port towards the other module, see Figure 3.4. The necessary information for that is available, namely the current orientation, the geometry of the module and the bearing of the other. The controller for rotation, which was introduced in Section 2.2, is used to accurately rotate the module such that the desired ports face each other. Afterwards, modules continue to the next state (approach and expect).

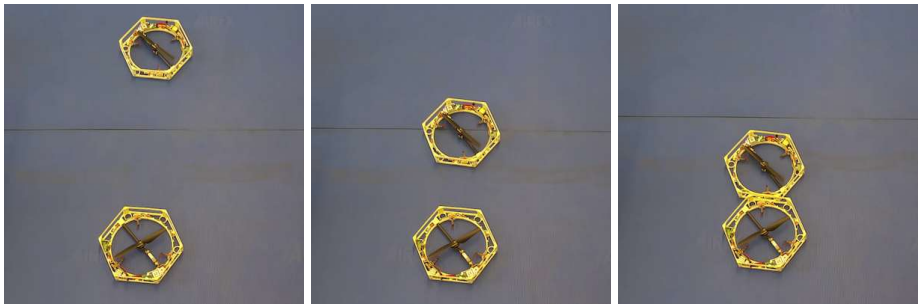


(a) (b)

Figure 3.4: Two modules are shown during the orientate state. (a) After modules have sensed each other they will be randomly orientated towards each other. (b) We can see the same modules after they orientated the desired port towards each other. Due to some errors in the measured quantities modules may not be perfectly aligned.

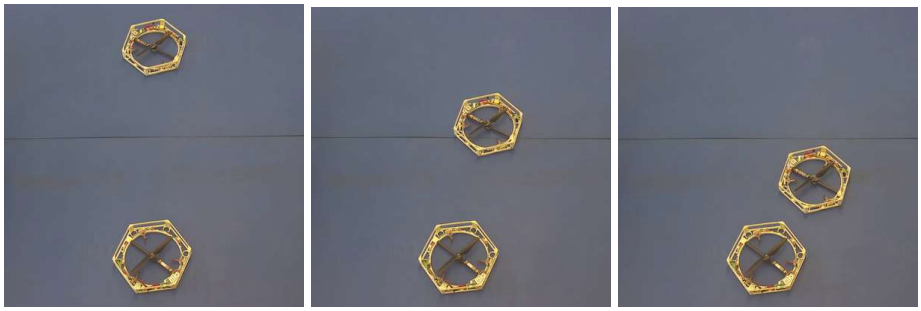
Approach and expect state

Depending on the unique ID of the modules, the one with the lower ID will move towards the estimated direction of the other one. The second one will not move and simply wait for the first one to reach it, see Figure 3.5. The reason why it was decided that only one of the two is approaching is related to the case when multiple modules pair up (Section 3.2). During this time both modules turn on the collision detection mechanism, see Section 2.5. Therefore, modules can sense when they have hit another object. They will then assume the object they collided with is the other module and proceed to the next state.



(a) (b) (c)

Figure 3.5: Three excerpts in the approach and expect state are displayed here. (a) Modules have finished orientating towards each other. (b) The one with the lower ID (in the back of the picture) is approaching the module with the higher ID. Both are in collision detection mode. (c) The one module has reached the other one. Since they are able to detect the collision they know they have reached each other and the approaching one stops moving.



(a) (b) (c)

Figure 3.6: In the case shown here one module did not estimate the bearing of the other one accurately enough and therefore approaches in the wrong direction. (a) Modules have already finished orientating towards each other but the module in the back is still misaligned due to errors in the estimation of the bearing. (b) While approaching it becomes more obvious that the bearing was not sensed correctly. (c) The timeout of the state was triggered and caused the modules to stop. Otherwise, the approaching module would have continued until it hit a random object.

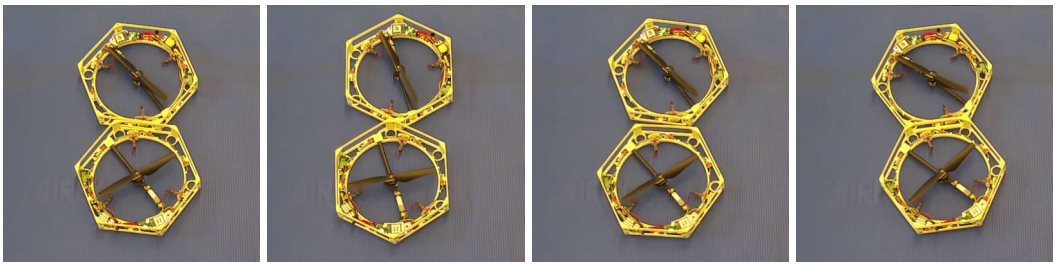
However, the case that modules do not reach other may arise and has to be considered. This can happen out of two main reasons:

- One of the two detects a false collision due to some obstacle or unexpected noise in the measurements.
- The bearing estimation of one module is so inaccurate such that it passes by the other one while approaching without touching it.

To account for those errors, the transition from this state to the next state (try to dock) will be solely triggered by the timeout of this state regardless whether a collision has been detected or not. Therefore the amount of displacement of the approaching module is limited in each iteration, even if it is approaching the other one correctly. This is due to the fact that there is no feedback on how accurately it is approaching once it has started moving. The case where one module inaccurately estimates the bearing of the other one can be seen in Figure 3.6. In the following state (try to dock) different actions will be applied depending on whether a module was able to detect a collision.

Try to dock state

In the ideal case (see Figure 3.5) the modules have reached each other and both have detected the collision. However, most likely modules will still be slightly misaligned and therefore they are not docked yet. Instead of immediately repeating the whole procedure, both modules will apply a sequence of motion to try to dock to each other, see Figure 3.7. This motion sequence consists of small rotations in both directions and short translational motions towards the other one in between each rotation. The sequence is only executed when a module detects a collision, whereas a module that has not detected a collision before will simply await the corresponding timeout of the state, again to keep actions between modules synchronized. This accounts for the case when modules have not reached each other yet, but also for the case only one module detects the collision. During this state (and actually during any other state as well) it is constantly checked whether a message could be received via the pins as described in Section 2.3. Once a message is received, modules know they have successfully docked to each other and will immediately terminate the process and go back to the idle state. In the event no message via the pins is received it is considered that they have not docked yet and they will then continue with the back up state. This may arise due to various reasons, namely because the motion sequence was not successful or modules have not reached each other yet.



(a) (b) (c) (d)

Figure 3.7: Modules apply a sequence of motion, consisting of alternating rotations in both direction and translations towards the other one, to dock to each other. (a) The picture shows the initial position of both modules when they have reached each other. (b) After the first time they have rotated and moved towards the other one, they are clearly more misaligned than before. This can happen since modules always start rotating in the same direction. (c) Modules have rotated back and will now again move towards each other. (d) The docking is successfully established and both modules stop moving.

Back up state

In case modules did not dock they need to repeat the whole procedure. For that both of them will back up approximately 10 cm by moving in the opposite direction of what was last estimated to be the bearing of the other one. Afterwards, modules will start over and go back to the find state.

3.2 Pairing Of Multiple Modules

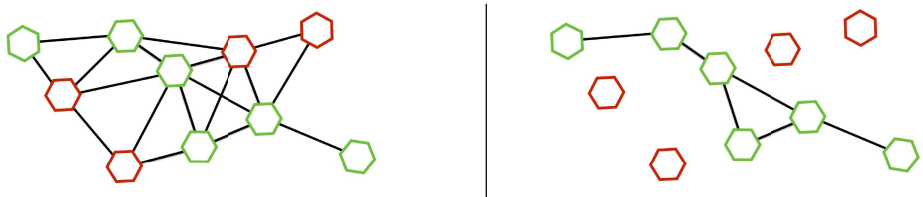
In the previous section we discuss how two modules can autonomously dock to each other. The following section proposes an algorithm to generalize this approach to an arbitrary number of modules, i.e. let modules estimate the bearing of their neighbors, decide on a partner to pair up with and finally dock to this partner in the before described manner. First, we make a few remarks on the algorithm in Section 3.2.1, then the way a configuration of multiple modules is modeled as a graph is reviewed (Section 3.2.2). Afterwards, we present the solution that was developed for letting modules pair up (Section 3.2.3). Several aspects of this approach are evaluated subsequently in Section 3.2.4.

3.2.1 Prerequisites

Similarly to the autonomous docking of two modules this algorithm should also adhere to the specifications proposed in Chapter 1. Unlike for the docking of two modules all of them should be met, which is accomplished in the following way: Homogeneity is achieved by letting all modules follow the exact same algorithm. As before an unique ID may be used to distinguish modules. A distributed approach is ensured since each module computes everything autonomously on board. The scalability is maintained by omitting any global knowledge of the structure. Therefore, the algorithm may accommodate for any arbitrary amount of modules. Finally, each module will apply the steps simultaneously to retain a parallelized approach.
Moreover, modules will use the IR–transceivers to exchange messages and communicate the decision process of who pairs up with whom.

3.2.2 Modelling Multiple Modules

Throughout the section we use various terms, which will be explained in the following: A neighbor or neighboring module, which was defined before in Section 2.3, refers to a module within the sensing range. A set of modules is described by the neighbor graph $\mathcal{N}$ where each node represents a module and each edge refers to the fact that the modules are neighbors, see Figure 3.8 (a). In case a path exists between every pair of modules the graph is considered connected. Further, when modules find a partner they are considered unavailable for other modules as partners. For this purpose a second type of graph, the availability graph $\mathcal{A}$, is defined, where a node corresponds to an available module and an edge refers to two available modules that are neighbors, see Figure 3.8 (b). Hence an edge of the neighbor graph $\mathcal{N}$ is not an edge of the availability graph $\mathcal{A}$ if at least one of the two modules is not available as a partner. $\mathcal{A}$ is therefore a subgraph of $\mathcal{N}$. For an arbitrary configuration of modules, $\mathcal{A}$ may be disconnected although the corresponding graph $\mathcal{N}$ is connected. It is noted that the terms connection and "to be connected" refer to the IR–connection between modules as defined for the corresponding graphs $\mathcal{A}$ and $\mathcal{N}$ but not to the physical state of two modules docked to each other. For that we use the term docking.



(a) (b)

Figure 3.8: Above, the neighbor graph $\mathcal{N}$ and the availability graph $\mathcal{A}$ of the same configuration are shown. Green modules are available and unavailable ones are marked red. (a) In the above configuration $\mathcal{N}$ is connected since there is a path between every pair of modules. (b) The corresponding availability graph $\mathcal{A}$ is also connected. It is to be noted that unavailable modules are not considered as a node of $\mathcal{A}$.

Perfect matching

Consider the following definition of a matching M [2]:

Definition 3.2.1. *Given a graph $\mathcal{G} = (N, E)$, consisting of nodes N and edges E, a matching M is a set of edges where any pair of edges does not share a common node. A node is called a match if it is an endpoint of an edge of the matching M.*

Therefore, a set of matches is a number of paired nodes or within this context nothing else than a number of paired modules that are going to dock to each other out of a larger entity of modules. Furthermore, for the application of pairing modules it is desirable to create as many pairs as possible. For that consider the following two definitions [2]:

Definition 3.2.2. *A maximum matching is a matching with the maximally possible number of nodes for a given graph $\mathcal{G}$. $\nu(\mathcal{G})$ denotes the number of matches in a maximum matching. There may be several maximum matchings of $\mathcal{G}$.*

Definition 3.2.3. *A maximum matching is called perfect if and only if $\nu(\mathcal{G}) = \frac{|N|}{2}$ where $|N|$ denotes the number of nodes. This can only be true when $|N|$ is even. For an odd $|N|$ one may define an almost perfect matching where $\nu(\mathcal{G}) = \frac{|N|-1}{2}$.*

In Figure 3.9 a maximum matching of two configurations is shown, whereas one of them is also perfect. In case of the pairing task the optimal solution would be the maximum matching of a given neighbor graph $\mathcal{N}$. This would guarantee that the maximal possible amount of modules pairs up. One may apply Tutte's Theorem [2] to find $\nu(\mathcal{G})$ and therefore check whether a perfect matching exists. The theorem is not stated here since it is beyond the scope of this report.

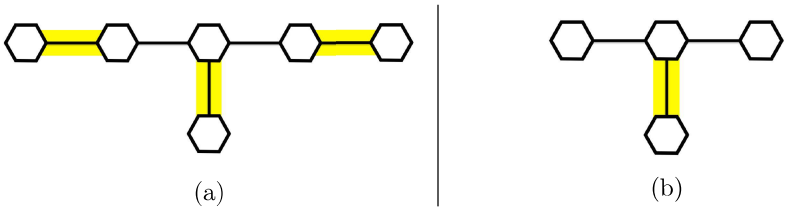


Figure 3.9: Two different neighbor graphs are shown with one maximum matching for each (matches are highlighted yellow). (a) For this configuration the maximum matching is perfect and a partner for each module can be found. (b) In this case only one match can be found and two modules are left without match.

3.2.3 Description Of The Algorithm

As stated before the goal is that an arbitrary number of modules sense each other, decide on a partner and then dock to the respective partner. Additionally, during the design of the algorithm to decide on a partner we considered the following aspects as well:

- The solution of the algorithm should converge and be unique since it is supposed to be implemented in a distributed fashion.
- It is desirable that the algorithm finds a maximum matching as this ensures that no additional modules may possibly pair up.
- Once modules have docked to their respective partner the new neighbor graph should still be connected. Otherwise pairs cannot communicate to each other anymore.

Taking into account the stated specifications as well as the prerequisites from Section 3.2.1 a module a_i bases its decision for its desired partner according to the following rules: Among all

of its available neighbors module a_i will choose the one that has the least available neighbors. In case several modules have the same amount of available neighbors, a_i choses the one with the least amount of total neighbors and out of those, if there is again more than one, it picks the one with the lowest ID. Algorithm 3.2.1 states the decision process in a more formal way and Figure 3.10 provides an example of such a decision process. The first two criteria were chosen to ensure that modules with less neighbors, i.e. with less possibilities to pair up, get paired up first and are therefore less likely to be left over when compared to a random decision process. The third criterion (ID) ensures that the behavior of the algorithm remains deterministic and the choice for the desired partner is always unique, which is of advantage when the algorithm is implemented in a distributed fashion. After deciding on a partner, it is checked whether two modules have chosen one another. In this case, the pairing is confirmed and the modules will become unavailable for others. When there are unpaired neighboring modules left the algorithm is executed again until no more pairs can be found. For modules that are unpaired but have no more available neighbors the algorithm will not be executed again since no partner can be found. They are going to remain without partner.

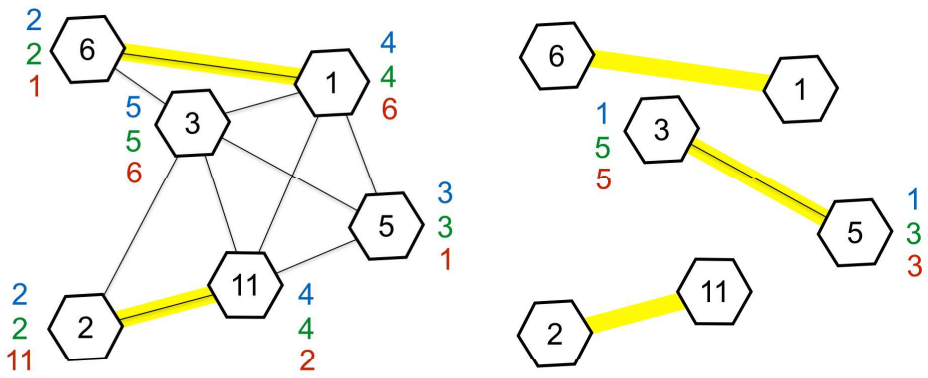
Algorithm 3.2.1 Choose desired partner of module a_i

$A(a_i)$: set of available neighbors of module a_i
$|T(a_j)|/|A(a_j)|$: amount of total/available neighbors of module $a_j \in A(a_i)$
$ID(a_j)$: unique ID of module a_j
function ChooseDesiredPartner
    $ID(a_p) \leftarrow ID(a_1)$ ▷ initialize desired partner (subscript p) with any a_j
    for all $a_j \in \mathcal{A}_i$ **do**
        if $|A(a_j)| < |A(a_p)|$ **then**
            $ID(a_p) \leftarrow ID(a_j)$
        else if $|A(a_j)| = |A(a_p)| \wedge |T(a_j)| < |T(a_p)|$ **then**
            $ID(a_p) \leftarrow ID(a_j)$
        else if $|A(a_j)| = |A(a_p)| \wedge |T(a_j)| = |T(a_p)| \wedge ID(a_j) < ID(a_p)$ **then**
            $ID(a_p) \leftarrow ID(a_j)$
        end if
    end for
    return $ID(a_p)$
end function

Only the module with less total neighbors out of each pair is moving towards its partner to increase the likelihood that the neighbor graph $\mathcal{N}$ stays connected once modules dock to each other. The other one will remain on the spot. Algorithm 3.2.2 provides more detailed information on how modules decide on which partner is approaching. The intuitive idea behind this is that modules with more neighbors are thought to be closer to the center while modules with less neighbors are thought to be more towards the convex hull of the graph. Letting modules move away from the convex hull closer to the center is supposed to ensure that the neighbor graph remains connected.

Algorithm 3.2.2 Decide if module a_i approaches or expects its partner

$|T(a_p)|$: amount of total neighbors of partner ($ID(a_p)$)
function DecideOnApproach
    $approach \leftarrow false$
    if $|T(a_p)| > |T(a_i)|$ **then**
        $approach \leftarrow true$
    else if $|T(a_p)| = |T(a_i)| \wedge ID(a_p) > ID(a_i)$ **then**
        $approach \leftarrow true$
    end if
    return *approach*
end function



(a) (b)

Figure 3.10: The pairing algorithm is executed for the above configuration. The column of numbers next to each module denotes the following: The amount of available neighbors (blue), the amount of total neighbors (green), the desired partner (red). The number inside a module corresponds to the ID and the connection of partners is marked yellow. (a) The availability graph $\mathcal{A}$ in the first iteration is shown. Two pairs can be found and two modules do not find a partner. (b) Since $\mathcal{A}$ still possesses a connected component, i.e. modules 3 and 5, they may execute the algorithm again. They are the only ones left available, therefore they choose each other.

Implementation on the DFA

The implementation of the algorithm is based on the IR–communication during the find state that is used to exchange the necessary information between neighbors. To ensure the algorithm converges in time modules now rotate longer (900°), see Section 4.2 for more information on why this is necessary. The following steps are applied during the extended find state:

1. Modules first rotate for 180° such that each neighbor is sensed at least once.

2. Each module has now knowledge of how many total and available neighbors it has with both numbers being equal in the beginning since no one has paired up yet. Next, they start sending out the amount of their total and available neighbors, which they will continuously do until a partner is found.

3. Once a module a_i has heard back from all its neighbors and therefore possesses information about their neighbors, namely the amount of available and total neighbors, the module may apply the pairing algorithm. The ID of the desired partner a_j will be sent out subsequently.

4. When a_i has received a message from a_j stating the partner module a_j desires a_i can decide whether there is a match or not. If both modules a_i and a_j have chosen each other the pairing for them is finished and both modules will from now on communicate that they are unavailable as partners.

5. In case there was no match, a_i will repeat the procedure starting at step 3 until it either finds a partner or all neighbors of a_i are unavailable leaving it unpaired.

Messages are sent out repeatedly in a fixed interval and altered in the different stages as described above. It is to be noted that the above scheme is asynchronous apart from step 1 allowing modules to individually decide when they can proceed to the next step. This approach features several advantages when compared to a synchronous version: To synchronize modules, one could include a central device to carry out the task. This, however, contradicts the specification that the overall

task should be executed in a distributed fashion. Another possibility would be if modules communicated when they are ready to proceed, which does not meet the required scalability though since the complexity increases with the number of modules. Simply timing the transition to the next step is also a way to synchronize the action but this method is not fault tolerant. Consider a situation where a module, out of some reason, has not received back any messages from its desired partner but it is required to already proceed to the next step. Moreover, the clocks of modules are not exact and, hence, drift. Under those circumstances an asynchronous approach is to be preferred over a synchronous approach.
Once modules have decided on a partner, they follow the state machine described in Section 3.1 to dock to their partner. The bearing is still estimated in the find state as for the case of two modules, only in the general case with multiple modules the pairing algorithm is executed in parallel. When modules did not successfully dock in the first place, they need to back up and go back to the find state. They will not change their partner anymore but simply estimate the bearing of the already found partner and then try to dock to it again. Figure 3.11 shows a set of six modules in the same configuration as in Figure 3.10 from the beginning until the moment they have docked to each other.

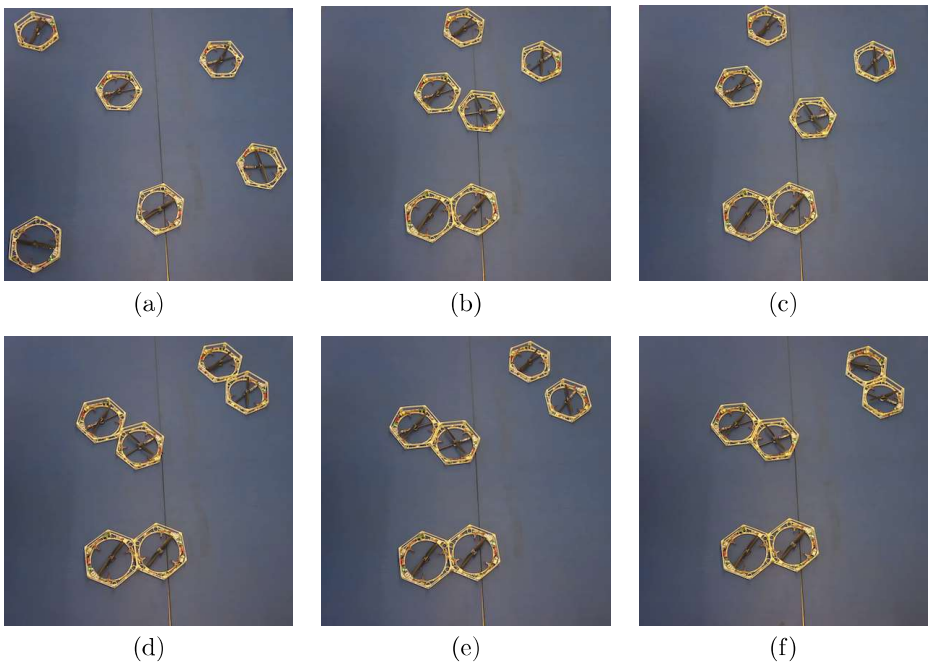


Figure 3.11: A set of six modules is shown during pairing up and docking to each other. (a) The initial configuration can be seen. Modules first decide on a partner and estimate the bearing of their neighbors. (b) Each module has found a partner and approached the partner. The pair in the front is already docked. The other pairs are going to back up to repeat the procedure. (c) Modules are again in the find state but this time they will solely estimate the bearing of their partner. (d) Both remaining pairs approached each other a second time. (e) The pair in the middle was able to successfully dock, whereas the pair in the back needs to start over a third time. (f) The last pair docked and the final configuration is achieved. Each module has paired up.

3.2.4 Discussion Of The Algorithm

In the following, the performance of the pairing algorithm is examined more closely, discussing various configurations and how the algorithm handles the situation compared to an optimal solution, i.e. the maximum matching.

Convergence

To begin with, we investigate whether the proposed algorithm always converges to a solution for any arbitrary configuration $\mathcal{N}$. For this consider the following properties that guarantee that a solution is found every time, although it might not be optimal.

Lemma 3.2.1. *For any given availability graph $\mathcal{A}$ that is connected, the pairing algorithm is guaranteed to find at least one pair.*

Proof.
Let a_i be any node from a given availability graph $\mathcal{A}$, $T(a_i)$ the set of all neighbors of a_i and similarly $A(a_i)$ the set of available neighbors of a_i. The modulus of any set refers to the number of nodes in this set. Further, n denotes the minimum of available neighbors for any node,

$$n = min(|A(a_i)|) \quad \forall i\,.$$

The subset $\mathcal{A}_L$ of $\mathcal{A}$ describes all nodes with n available neighbors,

$$\mathcal{A}_L = \{a_i \in \mathcal{A} \mid |A(a_i)| = n,\, \forall i\}\,.$$

Out of $\mathcal{A}_L$ the node a_L is chosen that has the following characteristics:

$$\{a_i \in \mathcal{A}_L \mid |T(a_i)| < |T(a_L)|,\, \forall i\} = \emptyset\,,$$
$$\{a_j \in \mathcal{A}_L \mid |T(a_j)| = |T(a_L)|,\, \forall j\} \cap \{a_k \in \mathcal{A}_L \mid ID(a_k) < ID(a_L),\, \forall k\} = \emptyset\,.$$

From the specifications of a_L it follows that any $a_m \in A(a_L)$ chooses a_L as desired partner because a_L is the module that is most prioritized by the pairing algorithm. a_L will now also choose a partner $a_m \in A(a_L)$. It immediately follows that there must be a match and one pair is found.

□

The next step is to discuss what happens to the new availability graph after the first pair(s) is (are) found. The algorithm actually does not guarantee that the availability graph of the remaining available modules stays connected, see Figure 3.12 for such a counterexample.

Lemma 3.2.2. *It is not guaranteed that the availability graph $\mathcal{A}$ of a given configuration remains connected after an iteration although the initial availability graph was connected.*

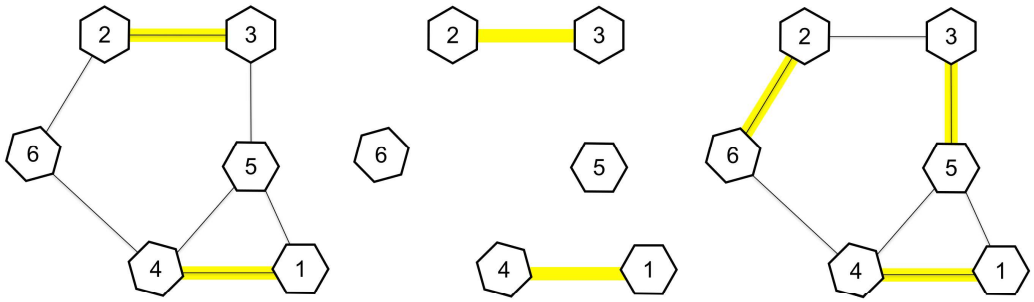


Figure 3.12: For the above configuration the pairing algorithm does not converge to a perfect matching although it exists because the availability graph after the first iteration gets disconnected. (a) During the first iteration two pairs are found. They are marked yellow. (b) The updated availability graph after the first iteration shows that the remaining two modules are not connected anymore, therefore no other pair may be found. (c) Actually, several perfect matchings exist, one of them is shown.

Nonetheless, it follows from the above lemmas that the algorithm converges to a solution whether or not it is the optimal solution.

Proposition 3.2.1. *The pairing algorithm will find at least one pair for a given availability graph and it will also find at least one pair for every connected component of the availability graph that results after the first iteration. This holds true for any number of iterations and any number of connected components resulting from a previous availability graph.*

Therefore, the minimal guarantee that can be given is that the algorithm does not create any deadlock situations and that it finds at least one pair for any given configuration.

Maximum matching

As visible in Figure 3.12 situations exist in which the algorithm does not find a maximum matching for a given configuration. In this sense (of finding the maximally possible number of pairs) the algorithm is not optimal. A solution that guarantees to find the maximum matching for a given graph actually exists, commonly called Edmonds's algorithm [3]. However, it was chosen not to rely on this algorithm for letting DFA modules pair up. The reason is that Edmonds's algorithm requires global knowledge of the structure, which violates the constraint of scalability.

Connection of graph after docking

All configurations considered so far remain connected even after modules docked to each other. However, this is not true for any configuration, see e.g. Figure 3.13. Each module finds a partner but then the connection of the docked pairs breaks up. Therefore, it cannot be guaranteed that all modules are able to communicate to each other afterwards. This does not impose any complications on the considered task, i.e. pairing modules, but it might be of interest for any subsequent action, e.g. creating bigger structures of modules.

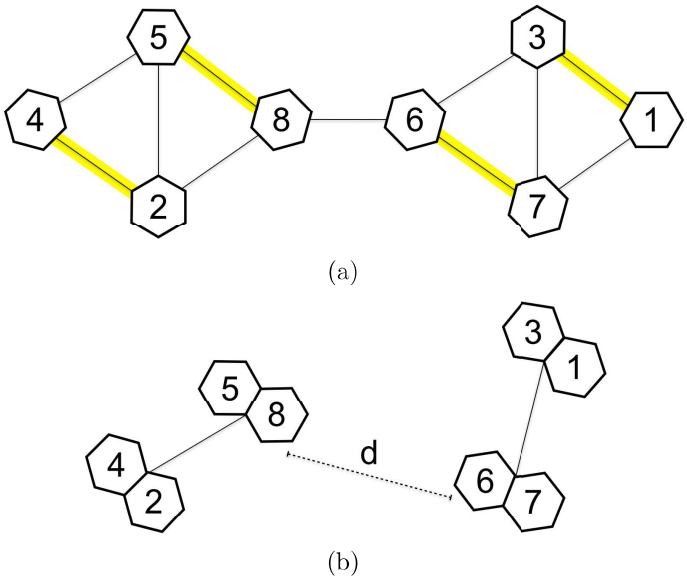
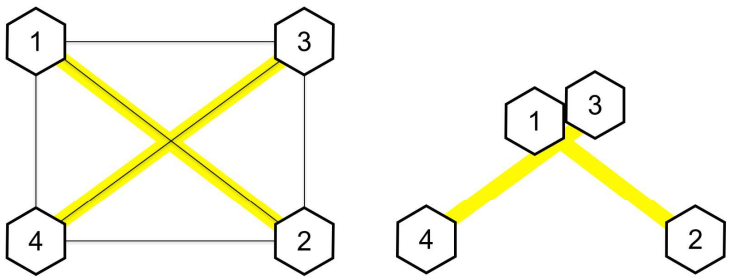


Figure 3.13: A configuration of modules is shown before and after docking. (a) The pairing results in the above pairs, which are marked yellow. (b) After docking to their partners, the new neighbor graph gets disconnected. Module 6 approaches its partner, module 7, and breaks up the connection to module 8. The distance d they are apart is larger than the sensing distance.

Crossing of modules

Another flaw in the pairing algorithm is of rather practical nature. Consider Figure 3.14 where each module sees all the other modules. Therefore, their decision for a partner is solely based on the ID and might turn out as as shown in Figure 3.14, where partners are across from each other. When the corresponding modules are approaching their partners, they will most likely hit each other causing them to detect a false collision and to initiate the motion sequence to dock. This will fail. After that, all modules will go back to the find state but since they block each other's vision they cannot sense their partners anymore. This will cause the process to be terminated leaving them undocked.



(a) (b)

Figure 3.14: Four modules in a configuration where each one is connected to all the others. (a) The pairs resulting from the pairing algorithm are across from each other. (b) The two modules that are approaching run into each other while they are moving since their paths cross.

Chapter 4

Experiments

In this chapter the performance of the pairing algorithm is experimentally analyzed and we discuss the results from pairing experiments that were conducted. Section 4.1 analyzes the performance of the state machine for docking two modules and in Section 4.2 the results from pairing experiments with six modules are evaluated.

4.1 Docking Of Two Modules

To analyze the docking of two modules 15 experiments were conducted with random initial orientation of the modules towards each other, see Table 4.1. In all cases in which modules sensed each other (11 out of 15) they were able to successfully dock to each other on the desired port. It took modules between one and three iterations to finish. The two main reasons that modules had to repeat the procedure were the partially inaccurate bearing estimation, also see Section 2.4.2, and the motion sequence for docking them that did not always result in a successful docking. Therefore, the approach is stable in case modules already have sensed each other. The reliability of the IR–communication again plays a significant role as for the bearing estimation.

4.2 Pairing Of Several Modules

The final experiments that were conducted investigated the performance of the pairing algorithm and the docking that follows it. Ten experiments with six modules were conducted in which modules were supposed to pair up and then dock to their corresponding partner, for the results see Table 4.2. In the Appendix A more experiments regarding the pairing may be found where four and five modules were used.
In five out of the ten experiments shown in Table 4.2 all six modules were able to pair up and dock to their corresponding partner. In one case all modules paired up but then one pair lost sight of each other when another module crossed their connection path. In the remaining four cases the algorithm was only able to find two pairs (three experiments) or one pair (one experiment). Designated partners mostly took one or two iterations to dock, sometimes it took them four or five iterations. As for docking two modules the reasons for that were either the inaccurate bearing estimation or the unsuccessful motion sequence during the docking attempt.
In the initial testing phase it was discovered that rotating 360° in the find state and communicating meanwhile as for two modules is not enough time for the pairing algorithm to converge. This is due to the unreliable IR–communication (Section 2.4.2 already investigated reasons for that). When modules look for a partner they need to receive back messages from all modules several times to come to a decision. This may take a while and therefore it was decided after further testing to increase the initial rotation phase where modules pair up to 900° to increase the probability the algorithm converges in time. For the experiments shown here modules rotated 900°.

	Initial distance apart	Success	Iterations	Time to finish	Additional comments
1	0.5 m	yes	1	20 s	
2	0.5 m	yes	2	46 s	
3	0.5 m	yes	1	21 s	
4	0.5 m	yes	1	22 s	
5	0.5 m	yes	3	1 min 11 s	
6	0.75 m	no	1	10 s to failure	sensed nothing
7	0.75 m	yes	1	21 s	
8	0.75 m	yes	2	46 s	
9	0.75 m	yes	3	1 min 10 s	
10	0.75 m	yes	1	18 s	
11	1 m	no	1	10 s to failure	sensed nothing
12	1 m	yes	2	46 s	
13	1 m	no	1	10 s to failure	sensed nothing
14	1 m	no	1	10 s to failure	sensed nothing
15	1 m	yes	2	46 s	

Table 4.1: The results from 15 docking experiments are shown. For all experiments it is stated how far modules were initially apart (center to center), whether docking was successful and how many iterations it took as well as the time to finish. In case of the unsuccessful events the time indicates the moment of the transition back to the idle state. The initial orientation in all experiments was random.

	Successful pairs	Iterations	Time to finish	Additional comments
1	3	2 2 2	1 min 1 min 5 s 1 min 6 s	One collision was not detected.
2	3	1 2 2	40 s 1 min 5 s 1 min 5 s	
3	2	2 2 2	1 min 1 min 18 s 1 min to failure	One collision was not detected. Pair 3 lost sight of each other because another module crossed their path.
4	2	2 4 1	1 min 5 s 1 min 53 s 24 s to failure	Algorithm did not find perfect matching; only two pairs were found.
5	1	1 1 1	42 s 24 s to failure 24 s to failure	Algorithm did not find perfect matching; only one pair was found.
6	2	1 4 1	42 s 1 min 53 s 24 s to failure	Algorithm did not find perfect matching; only two pairs were found.
7	3	1 1 2	41 s 41 s 1 min	
8	3	2 2 5	1 min 4 s 1 min 5 s 2 min 12 s	
9	3	1 1 2	41 s 42 s 1 min 5 s	
10	2	1 1 1	40 s 42 s 24 s to failure	Algorithm did not find perfect matching; only two pairs were found.

Table 4.2: Ten experiments with six modules that were supposed to pair up were conducted. Each row corresponds to one experiment and states how many pairs successfully docked to each other, how many iterations it took each pair as well as what was the finishing time. Reasons why not all modules paired up and further comments can be found in the last column.

Chapter 5

Conclusion

This thesis proposed a method to let multiple modules from the Distributed Flight Array pair up and afterwards dock to the respective partner. Modules decide on a partner according to the information they acquire from their neighbors, namely the amount of their total and available neighbors, and choose the module with the least available neighbors as their desired partner. If the other one wants to pair up too, the pairing is confirmed and modules will dock to each other. Otherwise they will continue looking for a partner until they have found one or no more modules are available. The first step towards docking to the partner is to estimate the bearing of it, which happens simultaneously to the pairing. For that modules rotate while listening to incoming IR–messages. In the moment they are line–of–sight and receive messages the current orientation angle acquired from the gyroscope is used to measure the bearing. Several measurements may then be averaged to gain an estimation. The IR–transceivers are also used to communicate the decision process. Afterwards, partners orientate towards each other and approach each other, whereas only the one with less total neighbors out of the two partners approaches the other. This measure was taken to keep all modules connected even after docking. A collision detection mechanism tells modules when they have reached each other and they are then going to try to dock to each other. If anything fails, modules repeat the procedure.

Once modules detected each other and paired up the method to let them dock proofed to be fully stable in all conducted experiments. However, the theoretical and experimental analysis of the pairing algorithm shows that there are still some unresolved issues to be tackled in the future. One main problem emerged from the unreliability of the IR–transceivers whose intervals of mutual visibility tend to be very irregular. This poses a huge complication on the implementation of the pairing algorithm because modules are required to continuously exchange messages with all neighbors. Currently, the period of rotation during the pairing was simply raised to increase the probability the algorithm converges in time. A more proper solution would be of advantage such as a communication device that allows to exchange messages at any time. The hardware of modules is actually able to accommodate a WiFi–device that could be used to regularly exchange messages for the task of finding a partner, whereas the IR–transceivers could still be used to estimate the bearing.
Furthermore, the algorithm itself is as well still to be improved. On the one hand, it is not guaranteed to find the maximum matching of a given configuration. Therefore, in a worst case scenario multiple modules might be left over although a better or even a perfect matching exists. One possible solution, Edmonds's algorithm [3], was given but it relies on global knowledge of the configuration, which violates the criterion of scalability. On the other hand, it can neither be ensured that the entire configuration remains connected after modules dock to each other preventing them from any further coordination of their actions. In this context it would also be of interest to further investigate what failures can be traced back to flaws in the algorithm and what is due to the unreliability of the IR–communication.

Eventually, the final goal is to not only assemble modules in pairs but also in larger, arbitrary structures. Based on the current work a strategy similar to a building block strategy is envisioned where two pairs first build a sub–cluster of four modules and those build sub–clusters of eight and so on until the final structure is achieved. To carry out this task, an assembly planer is necessary since it then matters how modules dock to each other. Moreover, we will need to refine the method to estimate the bearing of others to account for any arbitrary sub–cluster of which the bearing must be estimated. Similarly, the procedure of aligning/orientating modules has to be reconsidered because the alignment process then includes a translational part, however, as stated before there is currently no feedback on translational motion of modules.

Appendix A

Additional Experiments

In this chapter further experiments that were conducted with the DFA are shown. All experiments are of similar type to what was presented in Chapter 4. Section A.1 shows more experiments when driving with a DFA module, then the second set of experiments for the collision detection is discussed (Section A.2) and finally more pairing experiments are shown in Section A.3.

A.1 Drive Experiments

Figure A.1 shows the results from five experiments where the module was supposed to rotate 90°. It can be seen that the module does not always rotate too far as for the case of 360°. Also, the error in the onboard estimation of the orientation angle is below the average error after rotating 360°. This further reinforces the assumption that the gyroscope starts drifting during rotation. Since modules rotated less in this set of experiments, the error is also less. Note that the error is again almost identical in four out of five cases and shifted by about 2° in the fifth case.
The second set of five experiments presented here is shown in Figure A.2 where modules again were supposed to translate for 1 m but in a different direction (y instead of x). The results are very similar to what could be observed in the other experiments.

	Final orientation angle [°]	Error to desired orientation angle [°]
1	87.69	−2.31
2	89.75	−0.25
3	90.24	0.24
4	89.67	−0.33
5	91.86	1.86

Table A.1: The final orientation angle and the error to the desired orientation angle for the conducted experiments is shown (ground truth data). In all experiments the module was supposed to rotate 90°.

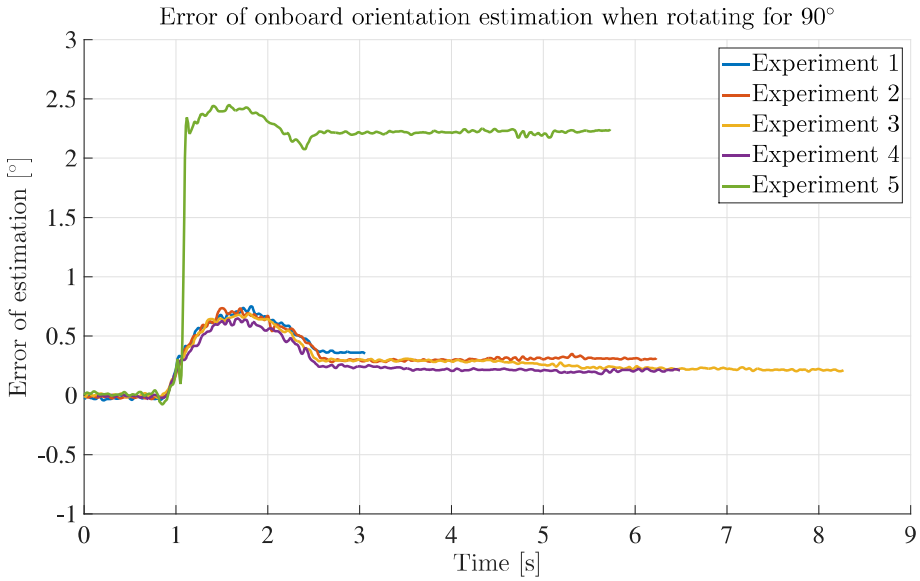


Figure A.1: The onboard estimation is compared to the actual orientation and the error is evaluated. In each experiment the module is supposed to rotate 90°. A positive error corresponds to an underestimated angle.

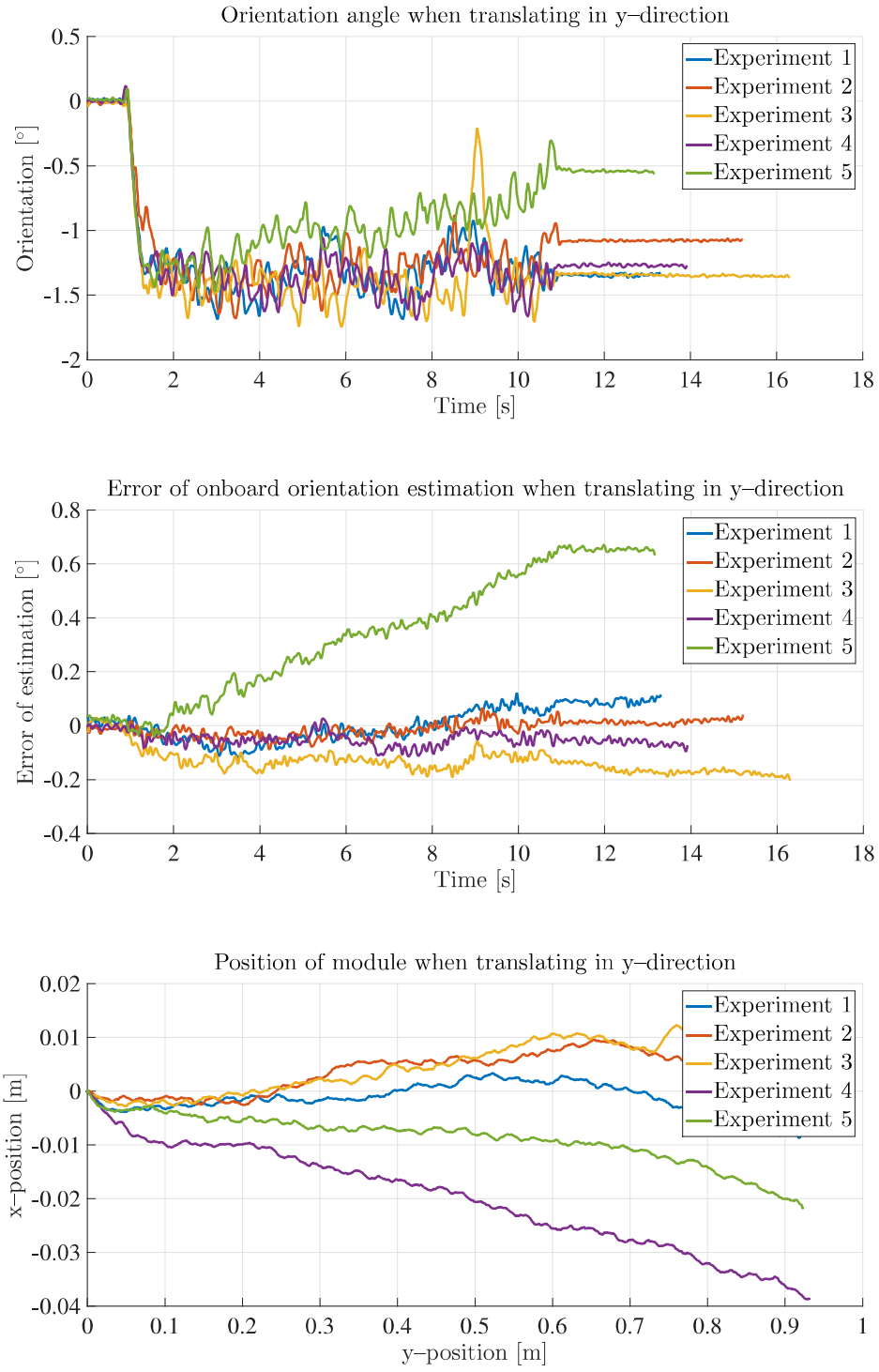


Figure A.2: In all five experiments shown above the module was supposed to move 1 m in y–direction (body frame) without rotating. The initial position and orientation was taken to be 0. In the top plot the orientation of the module during the experiment is shown. The break in the curve corresponds to the moment when the module starts moving. In the middle the error of the onboard orientation estimation is shown and the lower plot displays the position of the module.

A.2 Collision Detection

The second set of collision experiments is shown in Figure A.3. All collisions could be detected and there was no false detection.

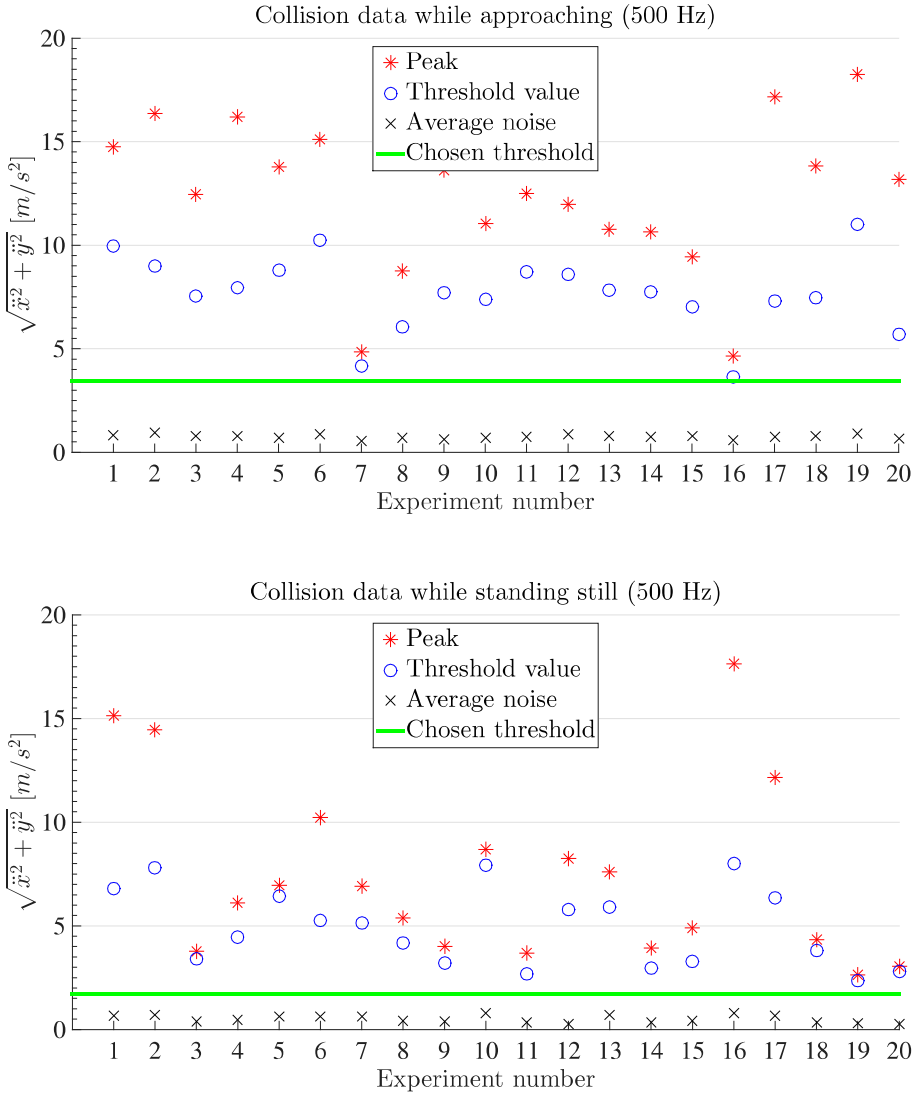


Figure A.3: For each experiment the peak acceleration (norm of x and y acceleration) is shown as well as the third highest value (threshold value) and the average noise. It is distinguished between an approaching module (upper plot) and a module that is standing still when the collision occurs (lower plot). All threshold values were above the chosen threshold.

A.3 Pairing Experiments

Ten additional experiments were conducted where four modules were supposed to pair up and dock as well as five experiments with five modules to pair up, see Table A.2 and Table A.3. Results are similar to what was seen before. Another source of error that did not arise so far was twice a wrongly calibrated gyroscope. This prevented the module from accurately estimating the bearing of its partner and then driving towards the partner. Also one time the IR–communication was so unreliable that none of the modules found a partner. When the same experiment was repeated all modules did find a partner.

	Successful pairs	Iterations	Time to finish	Additional comments
1	2	2 2	58 s 53 s	
2	2	1 1	32 s 34 s	
3	2	2 2	56 s 57 s	
4	2	1 3	33 s 1 min 22 s	
5	0	1 1	28 s to failure 28 s to failure	No pair was found due to bad IR–communication.
6	1	2 2	58 s 50 s to failure	Same setup as experiment 5 but two pairs were found this time. Pair 2 did not dock due to a wrongly calibrated gyroscope.
7	1	4 3	1 min 45 s 1 min to failure	One collision was not detected. Modules crossed each other and one pair lost sight.
8	2	1 2	34 s 1 min	
9	2	1 2	33 s 57 s	
10	1	3 3	1 min 20 s 1 min to failure	Two pairs were found but pair 2 did not dock due to a wrongly calibrated gyroscope.

Table A.2: Ten experiments with four modules that were supposed to pair up were conducted. Each row corresponds to one experiment and states how many pairs successfully docked to each other, how many iterations it took each pair as well as what was the finishing time. Reasons why not all modules paired up and further comments can be found in the last column.

	Successful pairs	Iterations	Time to finish	Additional comments
11	2	1 1	32 s 36 s	
12	2	1 1	33 s 33 s	
13	2	1 2	33 s 57 s	
14	2	1 1	33 s 39 s	
15	1	2 1	56 s 28 s to failure	Pair 1 docked on the wrong port after bad bearing estimation. No other pair was found

Table A.3: Five experiments with five modules that were supposed to pair up were conducted. One module should be left over therefore. Each row corresponds to one experiment and states how many pairs successfully docked to each other, how many iterations it took each pair as well as what was the finishing time. Reasons why not all modules paired up and further comments can be found in the last column.

Bibliography

[1] Stuart Cheshire and Mary Baker. Consistent overhead byte stuffing. *IEEE/ACM Transactions on Networking (TON)*, 7(2):159–172, 1999.

[2] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag Berlin Heidelberg, 4th edition edition, 2010.

[3] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[4] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[5] Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Autonomous self-assembly in swarm-bots. *Robotics, IEEE Transactions on*, 22(6):1115–1130, 2006.

[6] Roderich Groß and Marco Dorigo. Self-assembly at the macroscopic scale. *Proceedings of the IEEE*, 96(9):1490–1508, 2008.

[7] Philip Koopman and Tridib Chakravarty. Cyclic redundancy code (crc) polynomial selection for embedded networks. In *Dependable Systems and Networks, 2004 International Conference on*, pages 145–154. IEEE, 2004.

[8] Maximilian Kriegleder, Tejaswi Digumarti Sundara, Raymond Oung, and Raffaello D'Andrea. Rendezvous with bearings-only information and limited sensing range. 2014.

[9] Wenguo Liu and Alan FT Winfield. Distributed autonomous morphogenesis in a self-assembling robotic system. In *Morphogenetic Engineering*, pages 89–113. Springer, 2012.

[10] Sergei Lupashin, Markus Hehn, Mark W Mueller, Angela P Schoellig, Michael Sherback, and Raffaello D'Andrea. A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24(1):41–54, 2014.

[11] Nithin Mathews, Anders Lyhne Christensen, Rehan O'Grady, Philippe Rétornaz, Michael Bonani, Francesco Mondada, and Marco Dorigo. Enhanced directional self-assembly based on active recruitment and guidance. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4762–4769. IEEE, 2011.

[12] Rehan O'Grady, Anders Lyhne Christensen, and Marco Dorigo. Swarmorph: multirobot morphogenesis using directional self-assembly. *Robotics, IEEE Transactions on*, 25(3):738–743, 2009.

[13] Raymond Oung and Raffaello D'Andrea. The distributed flight array: Design, implementation, and analysis of a modular vertical take-off and landing vehicle. *The International Journal of Robotics Research*, page 0278364913501212, 2013.

[14] Miro Samek. *Practical UML Statecharts in C/C+*. Newnes, 2008.

[15] HongXing Wei, HaiYuan Li, JinDong Tan, and TianMiao Wang. Self-assembly control and experiments in swarm modular robots. *Science China Technological Sciences*, 55(4):1118–1131, 2012.

[16] Mark Yim, Babak Shirmohammadi, Jimmy Sastra, Michael Park, Michael Dugan, and Camillo J Taylor. Towards robotic self-reassembly after explosion. *Departmental Papers (MEAM)*, page 147, 2007.