Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORTHESIS

Autonomous Docking with Optical Positioning

submitted by

Kolja Poreski

MIN-Faculty

Department Informatics

Work group : TAMS

Degree Course: Informatik

Matriculation number: 6044143

First Supervisor: Prof. Dr. Jianwei Zhang

Second Supervisor: Lasse Einig

Abstract

English

The robot this work refers to is a TurtleBot from Kobuki. In this work, an optical method for position determination should be used in order to create a new automated docking algorithm for this TurtleBot. This new docking approach should replace the old one, from Kobuki that still uses Infrared. Another goal is to answer the question whether optical position determinations are precise enough to be used permanently for autonomous docking. Finally this new approach should be compared to the old one, by using a benchmark designed for docking approaches.

German

Der Roboter, auf den sich diese Arbeit bezieht, ist ein Kobuki-Turtlebot. In dieser Arbeit sollte eine optische Methode zur Positionsbestimmung verwendet werden, um einen neuen automatisierten Docking-Algorithmus für den Kobuki Turtle-Bot zu erstellen. Dieser neue Docking-Algorithmus sollte den alten von Kobuki ersetzen, der immer noch Infrarot verwendet. Ein weiteres Ziel ist es, die Frage zu beantworten, ob eine optische Positionsbestimmung genau genug ist um dauerhaft für das Docking verwendet zu werden. Desweiteren sollte dieser neue Algorithmus mit dem alten verglichen werden, der die Infrarotsensoren verwendet. Hierfür wird eine Auswertung beider Algorithmen durchgeführt um die beiden Ansätze mit hilfe eines Benchmarks zu vergleichen.

Contents

1 Introduction

Autonomous robotic systems are slowly entering areas of business and research. They will become more and more important in households. An example of an assistant robot in the household is an automated vacuum cleaner. A key role in their development plays algorithms and procedures that describe what the robot should do. It becomes more and more important that the robot performs its work autonomously. A battery-operated robot, for example, can explore the landscape autonomously, and for that, it must be able to autonomously decides to go to his charging station in order to recharge there.

The robot, this work refers to, is a Kobuki turtle robot. It comes with integrated infrared sensors on the robot and on the docking station. These sensors are used for determination of the position during docking. But this approach is not sufficient for the robot to work autonomously. In practice, docking often fails, because the infrared sensors do not deliver exact positioning data to the robot. However, these infrared sensors are disturbed by ordinary daylight, and at group TAMS some sensors were manufactured imprecise. This are two reasons why the docking with the on board sensors have a high error rate. Furthermore the algorithm, which comes from Kobuki, is not able to start a new attempt, after trying one unsuccessfully. Thus the robot is not usable when he has to operate autonomously.

Because there is a Kinect on the Turtle-Bot we want to implement a docking that is based on an optical positioning using an optical fiducial mark such as Apriltags. This has the advantage that it is not necessary to upgrade the robot with extra sensors. With an optical positioning it is even possible to print the position mark on the docking station. Thus it is a very cheep alternative to other approaches like RFID or Infrared.

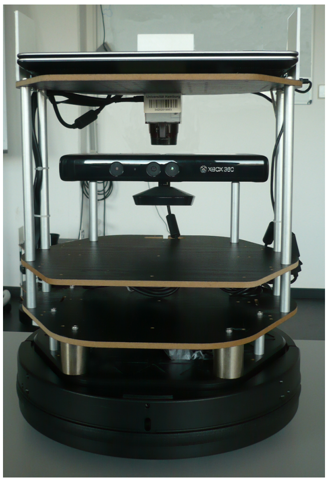
1.1 Objective

This work realizes an optical method for position determination and with a docking algorithm. Furthermore, the procedure described here is examined for accuracy.

The goal is to answer the question of whether optical position determinations are precise enough to be used permanently for autonomous docking. Furthermore, a node is to be implemented, that allows the robot to dock with the help of the Apriltags. Then this new approach should be compared to the old one, which uses the infrared sensors.

2 Description of the robot

There is a turtle robot of Kobuki, which can be programmed with the help of the ROS Framework. The ROS Framework offers the use the programming of the robot in Python or C/C++. *Nodes*, which are processes that run on the ROS system, can communicate with each other by using *ROS-messages*, that belong to a certain *topic*. A *topic* is formatted like a path (e.g. `mobile_base/sensors/imu_data`) and is unique in the ROS system. A *node* does publish a *message* to a certain *topic*, and another *node* can subscribe to this *topic*. Most information between the nodes are shared by using ROS messages and a topic. Thus all information that is needed to get information from a sensor is the *topic* where the messages are published and the type of the *message*.



(a) The Kobuki TurtleBot with the Kinect and the mounted laser sensor

(b) This figure shows a typical IMU. It is a PIXHAWK IMU that is described in [3].

(c) This is the laser low-frequency sensor, that is mounted on the TurtleBot.

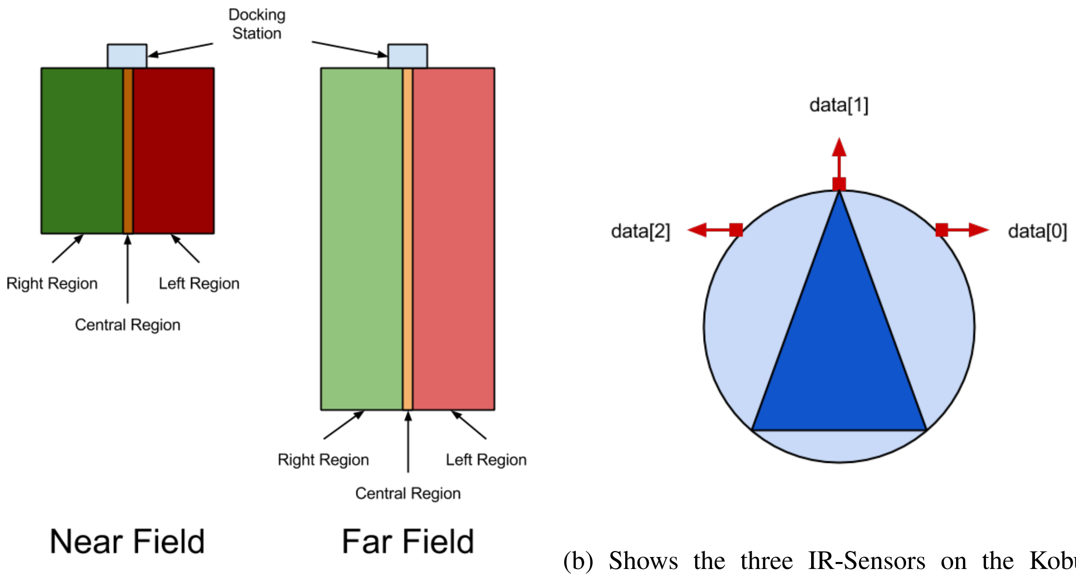
Figure 1: Shows the TurtleBot in a) an IMU in b) and the Laser-Sensor in c)

The robot has a Kinect (1a), a laser low-frequency sensor (1c), and an infrared interface, which is up to now used for the docking algorithm. The Camera on the Kinect [1] does have a field of view of 43° vertical and 57° horizontal. The videos taken from the Kinect has a resolution of 640x480 Pixel. The laser sensor as described in [2] is able to scan for obstacles. It can be used to master mapping, localization and navigation. Furthermore the Robot does have an IMU (=Inertial Measurement Unit) (1b) which publishes it's messages under the ROS-Topic
`mobile_base/sensors/imu_data`
using `sensor_msgs/Imu` messages. Most IMU's consist of different sensors. The PIXHAWK

Internal Measurement Unit described in [3] for example does have a *Gyroscope* to measure the rotation of the robot, a *Magnetometer* to measure magnetic flow density, an *Accelerometer* to measure acceleration and a *Pressure Sensor* to measure pressure. The list of sensors that are built in an IMU variates from model to model, but mostly these four mentioned sensors ca be found.

2.1 Kobuki's Docking Algorithm

Still today the robot uses a docking algorithm as described in [5]. As stated in [5] there are three IR-Receivers on the TurtleBot and three IR-Emitters on the docking station. In ROS the information from the IR-Receivers is available
under the topic `/mobile_base/sensors/dock_ir` in the form of a
`kobuki_msgs/DockInfraRed` message.



(a) Illustrates the tree different regions in front of the docking station. Each region is divided into near and far.

(b) Shows the three IR-Sensors on the Kobuki Turtle-bot

Figure 2: a) shows the different regions in front of the docking station, which come from the tree IR emitters on the docking station. b) shows the IR-receivers on the TurtleBot.

As shown in fig. 2a the IR emitters on the docking station divide the docking field into three Regions. A right region a left region and a central region. Furthermore it is distinguished between a near and a far field. Fig. 2b shows the three IR-Sensors, which are on the TurtleBot. One on the left, one on the right and one in the front.

The robot can be in one of the three regions. If the robot is in the central region, the robot simply has to follow the central region's signal until it reaches the docking station.
If the robot is placed in the left region as the starting position, it has to turn counter clockwise until the right sensor detects the left regions signal. At this position, the robot looks to the central region, so the robot just has to move forward in order to reach the central region. The robot has

reached the central region when the right sensor detects the central region. After the robot has reached the central region it just have to turn clockwise until the frontal IR-Sensor detects the central region.
The algorithm for the starting position at the right region is analog.

2.2 Problems of the Kobuki's Docking Approach

The Docking Algorithm which was previously shown should be replaced because there are three reasons why this approach does not work properly.
The first is, that the robot is not able to distinguish between different docking stations, because of using the infrared sensors, which are equally on all docking stations. If there are more than one docking station in the room it often happens, that the robot does not detect the nearest station but a station which is further away.
The second reason why the infrared approach does not work well is that the infrared sensors can be distorted by ordinary daylight. Thus the robot gets inexact values from the sensors, which often result in a failed docking.
The third reason is, that the IR-Emitters and IR-Sensors are sometimes badly manufactured and therefore inexact.

3 Important Terms

Terms like Odometry, dead-reckoning and visual odometry are often found in the literature. This chapter should explain the most important terms.

3.1 Odometry

In [6], Odometry is described as the process of estimate the position by using motion sensors. Odometry is the most widely used method for determining the position. Sensors can be any motion sensor like a Gyroscope or a speedometer. Mostly the term is used by wheeled robots and as sensor very often the revolutions taken by the wheels are meant. Because Odometry uses only motion sensors, which may have small errors in accuracy, the resulting position will become more and more imprecise after some time, because the errors from the sensors can accumulate.

3.2 Dead-reckoning

As mentioned in [7], dead-reckoning is the process of determining the current position by using a previously known position. Dead Reckoning is based on Odometry e.g monitoring the wheel revolutions to compute the offset from a known starting position. In many cases, dead-reckoning can reduce the cost of a robot system, because no extra sensors are needed. But there are two general error sources. On the one hand there can be systematic errors and on the other hand, there can be non-systematic errors. Systematic errors can be caused by kinematic imperfections e.g unequal wheel diameters. Systematic errors are regularly because the refer to properties of the robot itself. The second type of errors are the non-systematic-errors, that are caused by things like irregularities of the floor. These errors do not refer to properties of the robot.

3.3 Visual-odometry

As mentioned in [8], visual odometry is the same as odometry but instead of using different sensor types, it restricts you using a camera as a sensor. Whenever a visual input alone is used for positioning visual odometry is done.

3.4 Different Regions in the Docking-Area

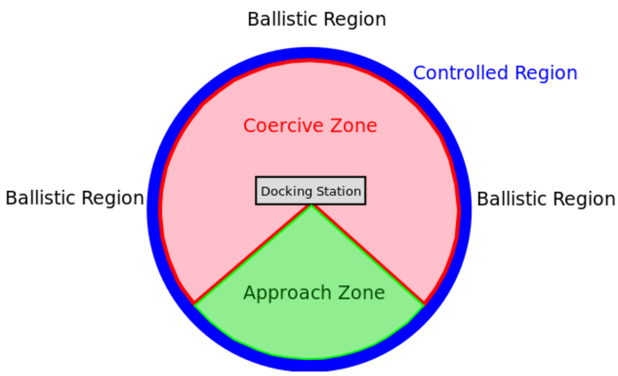


Figure 3: Docking regions as described in [9]

Before we talk about the docking algorithm we have to specify some regions which are important and mentioned in [9].

Ballistic Region
This is the area which is more than 2m away from the docking station.

Controlled Region
This is the area which is less than 2m around the docking station. This area can be divided into three different areas, the approach zone, the coercive zone and the frontal Zone.

Approach Zone
This is the zone where the robot sees the docking station or the Apriltag. The robot should be able to do the docking, when it is in the approach zone.

Coercive Zone
In this zone, the front of the docking station cannot be seen by the robot. Thus the robot has to do a repositioning in order to be in the approach zone. In this zone the robot is not expect to be able to do the docking.

Frontal Zone
In this work, a frontal zone is a zone where the robot must be in when it wants to drive straight forward into the docking station with only small corrections of the angle. This zone is although known as central zone.

4 Localization

In the literature, many different possible solutions for localization are mentioned. All these approaches can be distinguished by the use of the sensor system. In [10] the following approaches could be found.

- Infrared Sensors
  This approach measure the electromagnetic spectrum in the optical range of 1mm to 78nm wavelength.

- IEEE 802.11 RADAR
  This approach uses standard 802.11 network adapters and does the localization by measuring the signal strength.

- Ultrasonic
  The most approaches using ultrasonic sending an ultrasonic sound and measure the time-of-flight to gather the distance. This approach requires much infrastructure.

- RFID
  This approach uses radio-frequency identification.
  Like the IEEE 802.11 RADAR, it uses the signal strength to calculate the position.

- Optical
  Optical approaches use a camera and some kind of visual fiducial markers like Apriltags or QR-codes in order to determine the position.

Because not all these approaches are suitable for this work, only infrared, RFID and the optical approach is described more detailed, because for a new docking approach we are mainly interested in localization with respect to a local landmark.

4.1 Localization with Infrared

All infrared sensors can be put into two groups. The first are the active sensors, that work with self-emitted signals, the second are the passive sensors that react on an external signal source. Infrared works in a wavelength range of 850 nm to 50 µm. The Kobuki-TurtleBot uses passive IR-Sensors with IR-Emitters on the docking station. Because the daylight contains Infrared, these sensors can be distorted by normal daylight.

The following portable three-dimensional infrared local positioning system (IR-LPS) is mentioned in paper [11]. That system uses an IR-emitter and then triangulate the position of the robot by using two IR receivers. The system does have a range of 30m and an accuracy of 1cm at a range of 2m. The localization is done by determining the angle that the robot has relative to two points that are a known. Figure 4 shows the IR-emitter and two IR-receivers. The angles Ø, β and $\pi - \partial$ are known angles and DX is a known distance because the IR-receivers are put in place by the user.

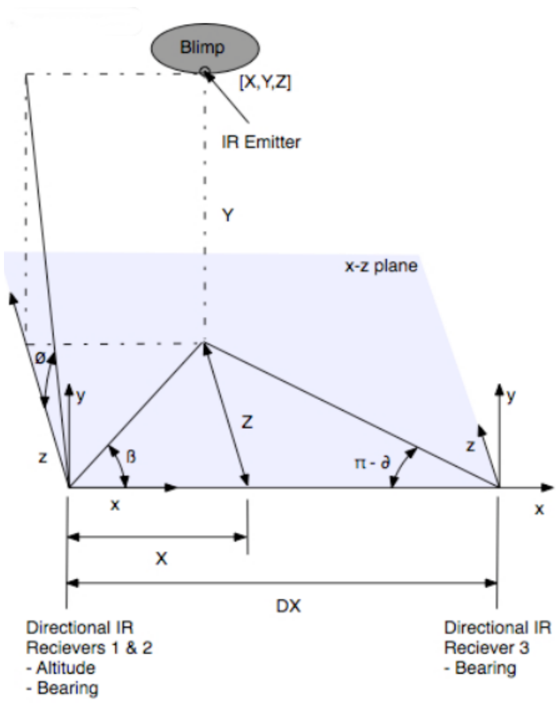


Figure 4: This figure from [11] shows the IR-Emitter on the top and two IR-Receivers placed on the x-z-plane. The Angles θ , β and $\pi - \partial$ are known angles and the distance DX is although known. The distance Z in the x-z plane has to be computed.

As described in [11] the following equations can derived form fig. 4:
I :$X = \frac{Z}{tan(\beta)}$
II :$(DX - X) = \frac{Z}{tan(\pi - \partial)}$
III:$Y = Z \cdot tan(\phi)$
and finally by subbing I into II

$$Z = \frac{DX}{\frac{1}{tan(\pi-\partial)} + \frac{1}{tan(\beta)}}$$

This approach comprises one IR-emitter and two IR-receivers, in order to localize the robot in the x-z plane.

4.2 Localization with RFID

The localization task can be solved by using RFID. A well-known RFID localization technology is SpotOn [10], which uses a three-dimensional location sensing based on the strength of the signal. Another System is the Spider System manufactured by RFCode [10] which uses a frequency of 308MHz. As described in [10], a RFID system consists of an RFID-Reader and RFID-Tags. The RFID-Reader reads the data from the tags. The RFID-Tags are available in an active and a passive form. The active tags operate with a battery, which powers a radio transceiver on the tag. The passive tags do not have a battery and no transceiver. It just reflects the signal from the reader and adds information by modulating the reflected signal. Active tags have more range than the passive once. The range is influenced by the angle subtended between the tag and reader. Both active and passive tags have unique hardware IDs, which makes it possible to distinguish between the different tags. Because the reader is not primarily designed to do localization tasks, there is often only one antenna in the reader, this makes the localization task difficult because since there is only one antenna it is not possible to detect changes while the robot is moving.
[12] describes the use of a dual directional antenna, which consists of two identical loop antennas, positioned to each other in a 90° phase difference. With such an antenna in the reader, it is possible to estimate the Direction of Arrival(=DOA), without scanning the environment. As

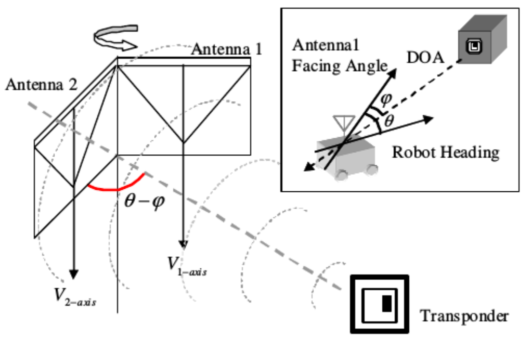
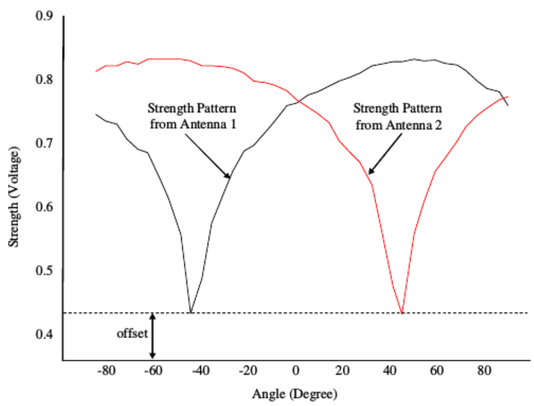


Fig. 1. Azimuth angle of the direction of arrival

(a) The RFID dual directional antenna, in which the voltage is inducted.



(b) The measured Voltage in antenna 1 and antenna 2

Figure 5: Both figures are from [12].a) shows the dual directional Antenna, which is used to make the DOA estimation. b) shows the measured Voltage in both antennas over the angle φ

described in [12], the induced voltage in the dual antennas can be computed by the following equations:

$$V_1 \propto \left| \frac{C \cdot S \cdot B}{r} \cdot sin(\theta - \varphi) \right|$$

$$V_1 \propto \left| \frac{C \cdot S \cdot B}{r} \cdot sin(\theta - \varphi) \right|$$

$$\upsilon_{12} = \frac{V_1}{V_2} = |tan(\theta - \varphi)|$$

S : surface area of the antenna

B : average magnetic flux density of the wave passing through the antenna

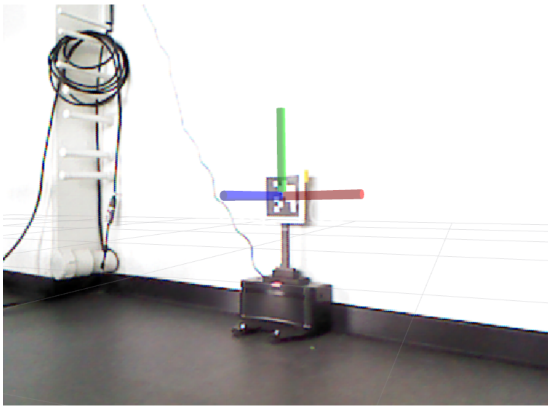
C : accounts for the environmental conditions

r : distance from the transponder

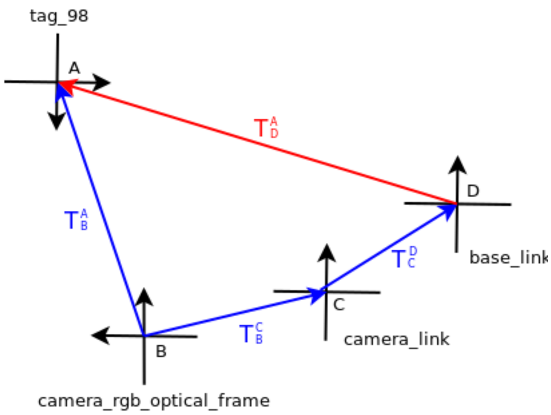
Thus V_1 , V_2 and θ can be measured in order to compute φ. As mentioned in [12] the accuracy of the DOA estimation using the directional antennas is within $\pm 4^\circ$ in an empty indoor environment. The RFID-based localization does have the advantage, that it is independent of the changes in illumination and it does not require an optical line of sight as many optical procedures. But the major disadvantage is scattering obstacles, that may distort the measurement of the voltages in antenna one and two. As stated in [12], it is almost impossible to include the whole scattering effect of signals in the obstacle cluttered environment.

4.3 Localization with Apriltags

After a small overview about some localization approaches, we should look closer into the Apriltags, because they will be used for localization in this work. Actually, Apriltags are similar to the well known QR-Codes, but QR-Codes have a different domain of applications. Because of that there are three disadvantages that QR-Codes would have if they would be used as landmark for localization. The first is that QR-Codes need a high Resolution, the second is that the user points the camera to the QR-Code thus QR-Codes does not need to be robust to rotation and the third disadvantage of QR-Codes is that often there is information coded into the Code. Apriltags are specially designed for localization, thus they do not have the three disadvantages, which means they do not need a high resolution, are robust to rotation and they code only necessary information into the code.
The procedure of the localization is done in two steps. The first is the detection of an Apriltag and the second is to calculate the position. The detection is the most CPU-intensive part. The Apriltag used in this work gives us a `poseStamped`. A `poseStamped` consist of a header and a pose. In the header, the frame_id and a timestamp is given and the pose consists of a point and a quaternion. This point gives the position relatively to the `camera_rgb_optical_frame` and the quaternion gives the relative rotation to the `camera_rgb_optical_frame`. Thus we can construct a transform T_B^A from Apriltag to the `camera_rgb_optical_frame`. As fig. 6b shows we



(a) This figure shows the detected Apriltag. The colored coordinate system (x=red,y=green and z=blue) is the coordinate system of the Apriltag.



(b) This figure shows all the frames that are involved by the computation of T_D^A.

Figure 6

can compute the transform T_D^A from the base_link D to the Apriltag A by the following equation.

$$[(T_B^A)^{-1} \cdot T_B^C] \cdot T_C^D = T_A^D \tag{1}$$

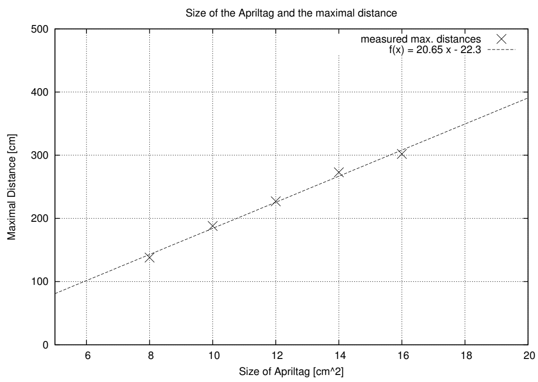
$$(T_A^D)^{-1} = T_D^A$$

4.3.1 Size of the Apriltag

On the one hand, the Apriltag should be small as possible, on the other hand, it must be big enough to produce sufficient detections in the required range of 2m. The luminosity does not have an influence on the detection range, because of the Apriltag Package [13], does a binarization of the image by using multiple filters. The size of the Apriltags is measured with the white surrounding. The measurement of the coherence between the maximal distance from the Apriltag and the size of the tag is detected experimentally. Therefore the robot was places in the frontal zone. Then the robot had to detect the Apriltag. If it was possible to detect the tag, the robot was pushed further away until there was not detection anymore. The distance the robot had when the first problems with the detection occurred were taken as maximal distance and noted. This experiment was repeated with different sizes of the Apriltag. The results are shown in fig. 7a. The resolution of the camera was 640x480 Pixel.

Size [cm^2]	D_{max} [cm]
8	138
10	188
12	227
14	273
16	302

(a) The relationship between the size of an Apriltag and the maximal range of the Apriltag.



(b) The plotted values from Table a) and the linear regression graph
$f(x) = 20.63x - 22.3$

Figure 7

As can be seen in fig. 7b the coherence between the size of the Apriltag and the maximal distance is linear. The resulting function $D_{max}(A)$ can be computed using a standard least square error method. The relationship between max. distance and size of the Apriltag can be written as:

$$D_{max}(A) = 20.63\frac{1}{cm} \cdot A - 22.3\text{cm} \tag{2}$$

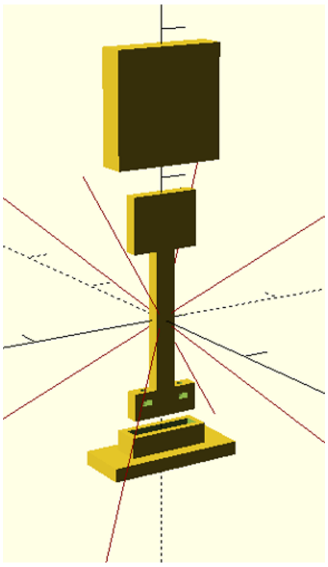
For the evaluation of the docking algorithm, a distance of 2m is required. According to equation

2 we need an Apriltag with a size of at least:

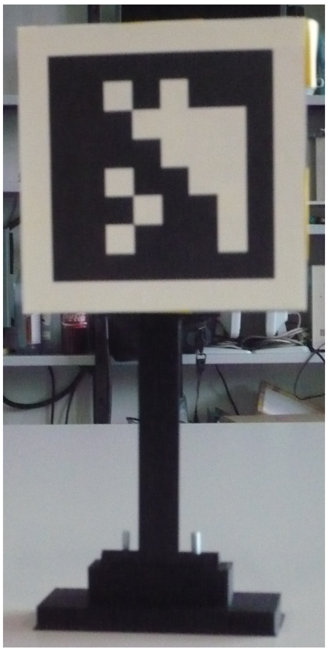
$$A = \frac{200cm + 22.3cm}{20.63\frac{1}{cm}} = 11.75cm^2 \Rightarrow 12cm^2$$

To be sure the Apriltag can still be detected at a distance of 2m a slightly bigger tag with $20cm^2$ is used.

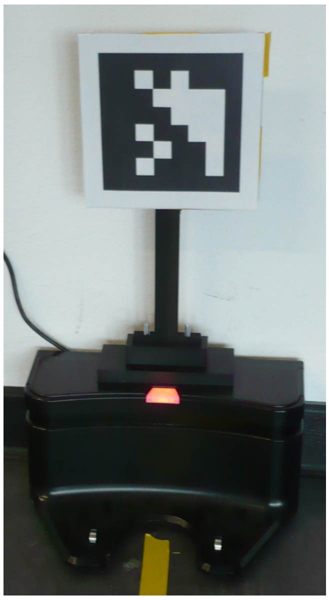
4.3.2 Tag Retainer



(a) The 3D-Model of the Apriltags retainer.



(b) Shows printed retainer.



(c) Shows the retainer on top of the docking station.

Figure 8

In order to mount the Apriltag on the docking station, a special retainer was designed using OpenSCAD and a 3D-Printer. As can be seen in fig. 8, the retainer consists of three parts. The fist part is the block, where the tag is put on. The second part is the bar, in order to variate the height. The third part is the foot. Each part of the, the retainer was designed to be replaceable, if there is a need of changing the height or the size of the tag. The foot is connected to the bar with two M4 screws and two hex nuts. The complete code of the 3D-Model is in the attachment at section 11.2.

5 Fundamental Functions

5.1 Angles

The yaw Angle α_{YAW} is an angle which states how two coordinate system are rotated to each other, when the z-axis is the rotation axis. When the x-axis of the robot's frame is rotated $\frac{\pi}{2}$ to the right, it looks in the same direction as the x-axis of the Apriltags frame. Thus the angle α_{YAW} is $-\frac{\pi}{2}$, if the robots frame B is not rotated. Then we can compute the position angle with $\alpha_P = tan(t_y/t_x)$.

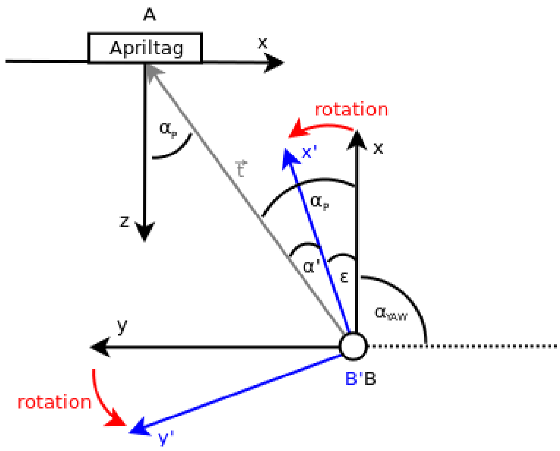


Figure 9: This figure shows the robot in front of the Apriltag. The robot is performing a rotation about the angle ε. The influence of the rotation and the angle α_P is shown.

If we rotate the robot the computation of the angle α_P has to be corrected. As shown in fig. 9 the angle ε is the rotation of the robots frame B to B'. The angle α_{YAW} can be expressed as the rotation angle between the x-axis of frame A and the x-axis from frame B. Thus $(-\frac{\pi}{2}) + (-\varepsilon)$ is the yaw angle between the frames A and B'. So we can write:

$$\varepsilon = -\frac{\pi}{2} - \alpha_{YAW}$$

As shown in fig. 9 the angle α_P can be found in the coordinate system of the robot because it is an alternate angle, when the z-axis of A and the x-axis of B are parallel. Furthermore α_P can be written as $\alpha^{'} + \varepsilon$. And $\alpha^{'}$ is in the frame B nothing else than $tan(t_y/t_x)$.
Thus we can write:

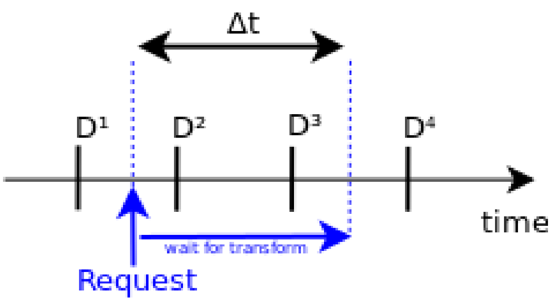
$$\alpha_P = \alpha^{'} + \varepsilon = tan(\frac{t_y}{t_x}) - (\frac{\pi}{2} + \alpha_{Yaw}) \tag{3}$$

With this equation we can compute α_P, when we only have the data from frame B', as long as

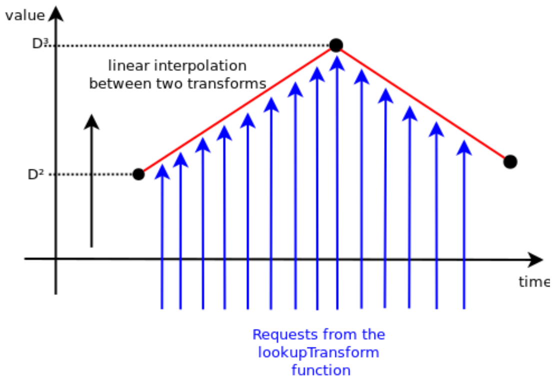
α_{YAW} is the rotation between A and B'. In a situation like the frontal docking where the robot has to drive directly to the Apriltag, it might be useful to ignore ε, because we want to have an angle, which becomes zero if the robot looks directly to the Apriltag in order to drive forward following the Apriltag. The angle $\alpha^{'}$ in fig. 9does have this property. The angle without the correction of ε is the docking angle α_D with:

$$\alpha^{'} = \alpha_D = tan(\frac{t_y}{t_x}) \tag{4}$$

In many use cases during the docking, we need an exact value especially when the robot moves during a measurement. In order to deal with outlier detection we have to average over the values. But if we use the function
`tf::TransformListener::lookupTransform(...)` or
`tf::TransformListener::lookupPose(...)`
from the tf library the values are not exact, because this function is doing an interpolation if an value is not available at the requested moment.



(a) The timeline shows the incoming detections of the Apriltag in TF and it shows the moment when the function lookupTransform(..) makes a request.



(b) The Graph shows the linear interpolation that is done with the values of the detected Transforms. Furthermore, it shows the requests that are made by TF are illustrated by the blue arrows.

Fig. 10a shows some occurring detections and the moment when the request with the `lookupTransform()` function is done. Furthermore, we can set the time tf has to wait for a detected transform if there is none at the requested moment. If there is a new Transform available, the tf Library does now a linear interpolation and gives back the interpolated value at the requested time. If we now assume that the `tf::TransformListener::lookupTransform(...)` function is in a loop in order to always refresh a value, we know there are many requests at a very high frequency. Like in fig. 10b illustrated we get an interpolated value very often. If we try to make the detection more exact, by averaging over 40 values in order to illuminate outliers. We have to take into

account that we cannot average over the blue requests in fig. 10b, because we would average over a linear interpolation.
The following figure 15 shows the result of averaging over these interpolations.

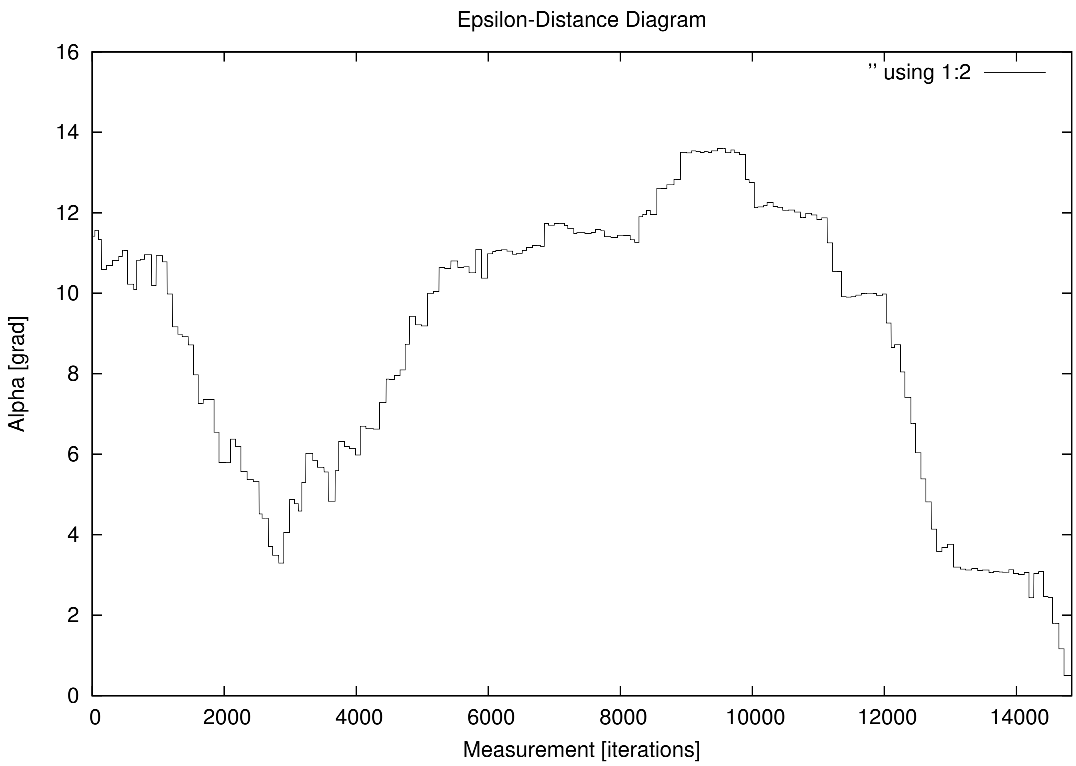


Figure 11: This graph shows the the measured angle α_P using the`tf::TransformListener::lookupPose()` function during the linear-approach, in which the robot drives to the ideal path by keeping the Apriltag in the visible field. This is an example in which the avarage of 40 values is made over the linear interpolation. Actually this Graph should be linear with a negative gradient.

All in all we cannot use the tf Library, because we want to average over the values to illuminate outliers.
Thus we have to write our own transform function, which computes the transform between the frames "base_link" and the Apriltag according to the equation 1. And which does wait for detections of the Apriltags, and calculates the transforms, when the new values arrive directly from the Apriltags node as `poseStamped` Messages. Instead of asking the system to give us a value at a certain point in time, we just wait when a new detection arrives and then we make the transform calculations.

```
void get_avg_position_angle(const apriltags_ros::AprilTagDetectionArray& msg)
    {
if(start_avg){
  apriltags_ros::AprilTagDetection detection ;
    int size = msg.detections.size();
  if(size != 1){
   ROS_ERROR("get_avg_position_angle must have exactly one apriltag");
  }else{
      apriltags_ros::AprilTagDetection detect = msg.detections[0];
      geometry_msgs::PoseStamped pose          = detect.pose;
      geometry_msgs::Pose p                    = pose.pose;
      geometry_msgs::Point point               = p.position;

  float x = point.x;  float y = point.y;  float z = point.z;
  tf::Quaternion optical_tag_quad   = p.orientation;
  tf::Vector3 *optical_tag_origin   = new tf::Vector3(tfScalar(x),
                                      tfScalar(y),
                                      tfScalar(z));
  tf::Transform *optical_tag = new tf::Transform(*optical_tag_quad,*
    optical_tag_origin);
  tf::Transform tag_cam      =  optical_tag->inverse() *
    _transform_optical_cam;
  tf::Transform tag_base     = tag_cam * _transform_cam_base;
  tf::Transform base_tag     = tag_base.inverse();
  float _x           = base_tag.getOrigin().x();
  float _y         = base_tag.getOrigin().y();
  float _z         = base_tag.getOrigin().z();
  float yaw        = tf::getYaw(base_tag.getRotation());
  float yy         = getAmount(_y);
  float alpha_dock = atan(_y/_x);
  float alpha_pos = (alpha_dock)-(M_PI/2 + yaw));
  if(!isnan(alpha_pos)){
      _avgPos.new_value(alpha_pos);
      _avgDock.new_value(alpha_dock);
      _avgPositionAngle = _avgPos.avg();
      _avgDockingAngle  = _avgDock.avg();
    }
  }
}
}
```

Figure 12: This code shows how the transform between the base_link and Apriltag is computed, then the function computes the position angle α_P and the yaw angle by calculating the mean of 10 values.

5.2 Moving forward

There is a fundamental need for a function that makes it possible to drive an exact distance forward. The easiest way to do that is a timing method because we do not need extra sensors.

Timing-method means we calculate the time we need to drive and stop the motor when this point in time has been reached. We have given the velocity v_c and the way we want to drive. If we assume a constant velocity we can compute the time we want do drive by $t = \frac{way}{v_c}$.
But in reality the robot first has to accelerate before it reaches the given velocity, thus the velocity is not constant. As fig. 13 shows there is an acceleration phase and after that we drive at nearly constant velocity v_c.
A good theoretical approximation for the behavior of the velocity is:

$$v(t) = -\frac{A}{t} + v_c \text{ with A as acceleration}$$

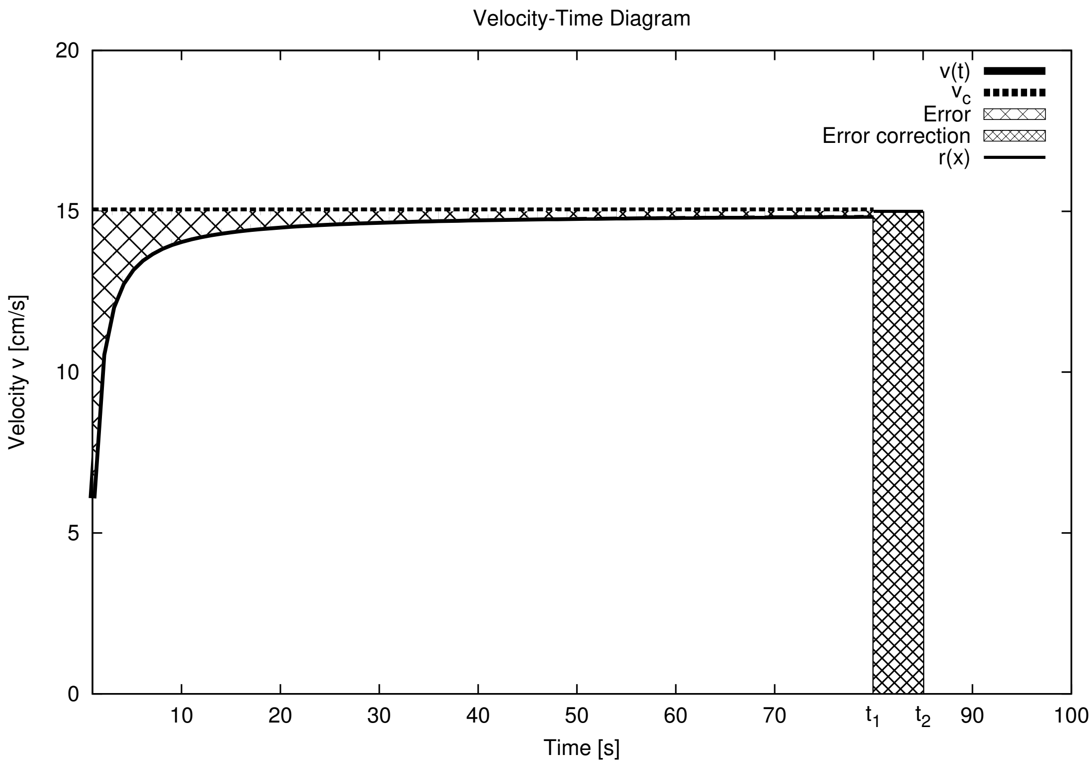


Figure 13: This figure shows the velocity-time graph, which shows the acceleration of the robot. In the first phase the robot accelerates until it has reached the stated velocity at $15\frac{cm}{s}$.The mistake made by the acceleration phase is illustrated by the Area A_1.

As shown in in fig. 13 , we first have an acceleration phase and after some time we drive with nearly the given velocity v_c. We can assume that the error made by assuming a constant velocity, is given by the area between the driven velocity v(t) and the constant velocity v_c. An area in the v-t-diagram is always a way. The error is calculated, by taking the difference between the driven way W_d and the expect way W_e.

$$E(t) = W_e(t) - W_d(t) = v_c \cdot t_1 - \int_0^{t_1} (v(t))dt = v_c \cdot t_1 - [-A \cdot ln(t) + v_c \cdot t_1]$$

$= A \cdot ln(t)$ with $t = \frac{W_e}{v_c}$ gives:

$$E(\frac{W_e}{v_c}) = A \cdot ln(\frac{W_e}{v_c}) = A \cdot ln(W_e) - A \cdot ln(v_c)$$

All in all, we have a logarithmic coherence between the error and the expected way. Because we want to measure the error we make a simplification on the equation which leads to:

$$E(W_e) = A \cdot ln(W_e) - B \tag{5}$$

The values for A and B will be determined experimentally because the function for v(t) is just an approximations, which does not contain specific terms for distributing forces like the friction. All distributing forces will be considered by determining A and B experimentally. Thus B will not be exactly the $A \cdot ln(v_c)$ from the theory. Once we have an error function we can calculate the error, we want to know how much longer the robot has to drive in order to compensate the error made by the acceleration. As shown in fig. 13 the robot will have reached nearly v_c at t_1. Thus he has a nearly constant velocity at that point in time and we know we have to drive $E = W_e - W_d$. All in all we have to drive $dt = \frac{E}{v_c}$ longer.
Now we have an equation to compute the additional time the robot has to drive in order to compensate the error made by the acceleration phase. But we still have to determine the parameters A and B.

To determine A and B we measure the error $E = W_e - W_d$, we use a constant velocity $v_c = 15\frac{cm}{s}$. The following table shows the expected way the robot should drive and the actually driven ways. After the measurement, we compute the average error and create an E-w-Diagram (fig. 14) in which the error is associated with the way (W_e).

W_e	25cm	50cm	100cm	150cm	175cm	200cm	250cm	300cm
W_d	25.0 23.45 24.85 23.45	44.65 44.35 45.85	— 93.25 88.05 88.25 87.85 88.75 88.85	119.85 133.95 140.85 132.85 133.35 134.15 134.35 33.85	157.55 157.55 157.55	180.35 181.35 180.75 180.25 180.95 179.35	229.65 229.55	278.35 278.25 277.25
∅ E	0.82	5.05	13.65	17.10	17.45	19.5	20.40	22.05
MSE	1.21	25.92	276.37	193.06	304.50	380.65	416.16	486.45

Table 1: The expected way W_e, the actual measured driven ways (W_d) and the average error ∅ E calculated for every W_e

Then we can calculate the values for A and B by using a standard least square method. This method gives us:

$$E(W_e) = 8.92 \cdot ln(W_e) - 28.35$$

and

$$dt = \frac{E(W_e)}{v_c} \tag{6}$$

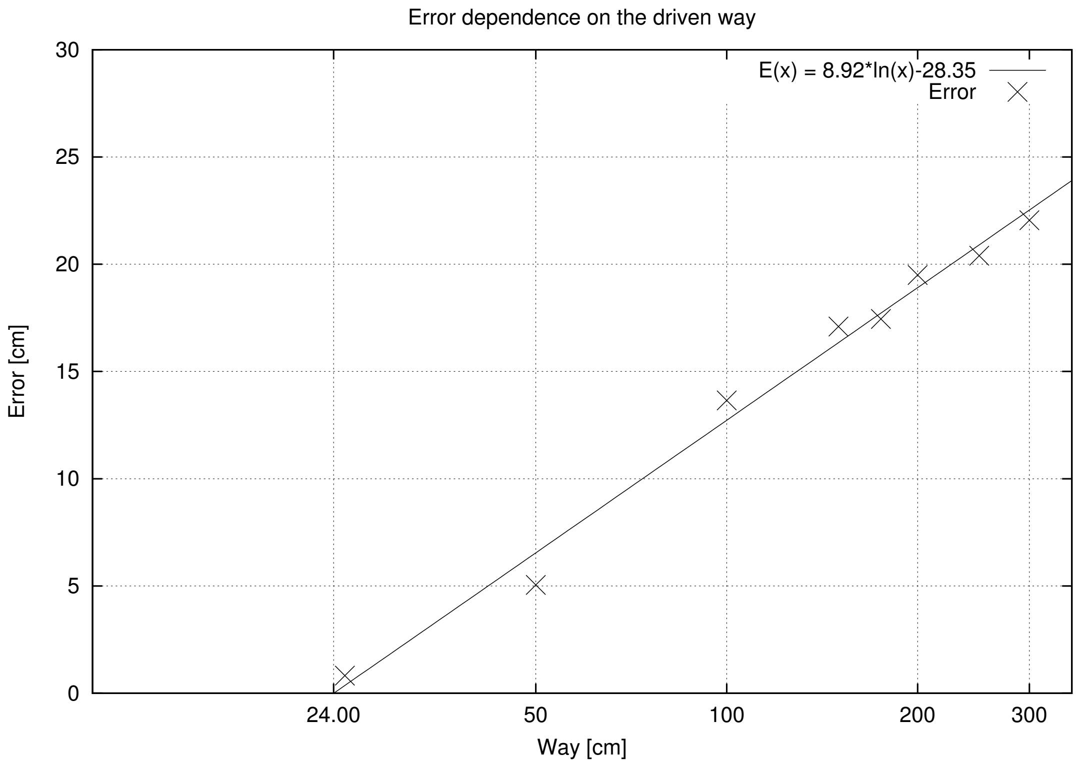


Figure 14: This diagram equation 7 for way > 24cm and it shows the measured actually driven ways with their errors. The x-axis does have a logarithmic scale.

The plotted average errors have a logarithmic correlation to the expected way. But if you look closer the small ways, e.g 1cm the found function does not seem to be correct, because $E(1cm) = 8.92cm \cdot log(1) - 28.35cm = -28.35cm$ this would mean that the robot has to drive backwards, in order to correct an error. Thus the found function is not correct for small distances. This happens, when the distance is so small, that the robot reaches the end, but did not reach the given velocity v_c. Thus the velocity (v_{end}) the robot has, in the end, is smaller than v_c. And then the equation 6 is not correct, it would be correct if we used v_{end} instead of v_c, but this is not possible because we do not know this velocity. We denote all distances as small that have a negative error, calculated by equation 5 with the found values for A and B. Thus $E(x) = 0 \Leftrightarrow x = 24.00cm$ is the boundary between small distances and normal distances. The equation works fine for distances bigger than 24 cm.

If we have to drive a distance under 24cm we set the velocity to a very slow value. The robot will reach v_c very quickly and the error that comes from acceleration will be very small. The same measurement as above was made, and this time a linear correlation between the error and small distances was found.

W_e[cm]	2	4	6	8	10	12	16	18
W_d[cm]	1.6 2.0 1.7 1.5	2.8 2.8 2.5 2.6	3.6 3.5 3.9 3.8	4.6 4.7 4.9 4.7	6.2 6.1 5.9 6.3	6.8 7.0 7.1 7.4	9.7 10.1 10.2 10.1	11.0 11.4 10.6 11.3
$\varnothing W_d$	1.7	2.675	3.70	4.725	6.125	7.075	10.025	11.075
$\varnothing E$	0.3	1.325	2.30	3.275	3.875	4.925	5.975	6.925
MSE	0.035	0.017	0.025	0.012	0.022	0.047	0.037	0.097

Table 2: The ecpected way W_e, the actual measured driven ways (W_d) and the average error $\varnothing$ E calculated for every W_d

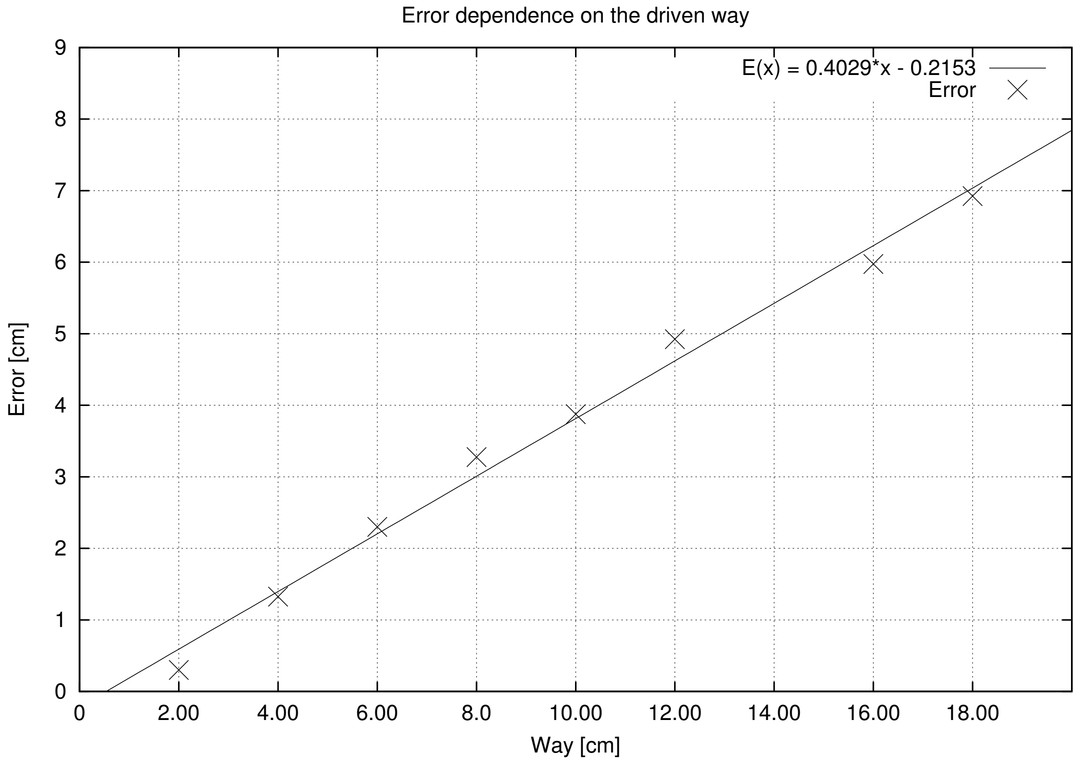


Figure 15: This diagram equation 7 for way < 24cm and it shows the measured actually driven ways with their errors.

Then we can calculate the values for A and B by using a standard least square method. It was found : $E(way) = A \cdot x + B$ with $A = 0.4029$ and $B = -0.2153$ If we put both equations for large and small distances together, we get:

$$E(way) = \begin{cases} 0.4029 \cdot way - 0.2153 & \text{for } way < 24 \\ 8.92 \cdot ln(way) - 28.35 & \text{for } way \geq 24 \end{cases} \tag{7}$$

With this equation for the error we can implement the following pseudo code.

Algorithm 1 DriveForward

```
procedure DRIVEFORWARD(distance)
    velocity = 0.15
    time_to_wait ← distance/velocity
    ERROR ← 8.92 · log(distance ∗ 100) − 26.515
    dt ← ERROR/(velocity ∗ 100)
    if td > 0 then
        time_to_wait ← time_to_wait + dt
    else
        velocity ← 0.05
        time_to_wait ← distance/velocity
        ERROR ← 0.395 · (distance ∗ 100) − 0.079
        dt ← ERROR/(velocity · 100)
        time_to_wait ← time_to_wait + dt
    startTime ← Time :: now()
    while (Time :: now() − startTime) < time_to_wait do
        drive_forward(velocity)
        Duration(0.5).sleep()
```

5.3 Turn by an Angle

It is also very important to be able to turn the robot exactly. Again we have two possibilities to stop the turning process when the stated angle is reached. The first one is a timing method, the second is using the IMU sensor to detect the angle. The IMU is a built-in sensor that can measure the angle the robot has turned. Thus we do not need the timing method because with an exactly measured angle, the movement will be more precise than using the timing method. The IMU gives us the angle in a range from -180° to +180°. When the IMU is initialized the angle the robot stands will be set to 0°. Because of that is is not possible to give an angle bigger than 180°. If an angle e.g 190° is given the sensor would have to stop at -170° because the measured value jumps from 180° to -180 and then it goes +10° further. Thus we first have to adjust the stop angle in order to be in the range of -180° and +180°.

```
ros::Subscriber sub2 = _n.subscribe("mobile_base/sensors/imu_data",
                    1,
                    &Docking::actual_angle,
                    this);

void actual_angle(const sensor_msgs::Imu& msg){
        _angle = tf::getYaw(msg.orientation);
}

void move_angle(float alpha_rad){

if(alpha_rad != 0.0){
    float goal_angle    = _angle + alpha_rad;
    int   N             = (int)getAmount((goal_angle / M_PI));
    float rest          = goal_angle - ((float)N * M_PI);

    if((rest > 0) && (N > 0) ){
        //This is nessesary because the imu is between -180 - 180
        goal_angle = (goal_angle > M_PI) ? -M_PI + rest : goal_angle;
        goal_angle = (goal_angle < -M_PI) ? M_PI - rest : goal_angle;
    }
    float rad_velocity = (alpha_rad < 0) ? -0.5 : 0.5;
    bool weiter = true;

    while(weiter){
        geometry_msgs::Twist base;
        base.linear.x    = 0.0;
        base.angular.z   = rad_velocity;
        _publisher.publish(base);
        //We wait, to reduce the number of sended TwistMessages,
        //If we would not the application crashes after some time.
        ros::Duration(0.5).sleep();
        float epsilon    = goal_angle - _angle;
        epsilon    = getAmount(epsilon);
        weiter = epsilon  > 0.1 ;
    }
}
}
```

Figure 16: The code shows the function which is used to turn the robot around the z-axis.

6 Docking Algorithm

In order to dock reliably, we have to divide the docking algorithm into five parts. The basic idea of the algorithm is very easy, first the robot drives somewhere in the approach zone. In the second step, the robot turns to the Apriltag. The third step consists of driving to the ideal line until the robot stands directly in front of the Apriltag. If Step3 does not position the robot exactly enough, Step4 may correct this by using a linear approach to the frontal zone. After standing in front of the Apriltag the robot can begin with the frontal docking.

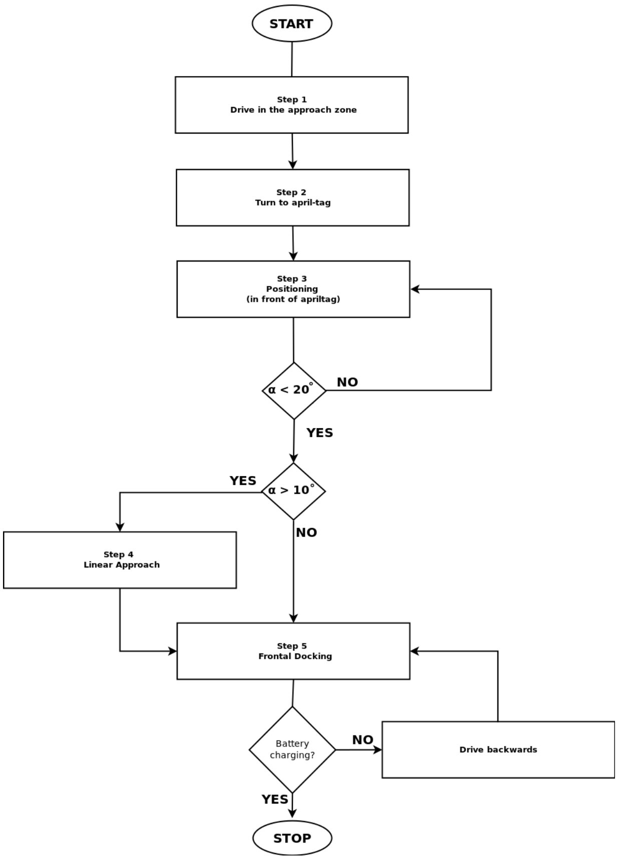


Figure 17: Flowchart of the docking algorithm with it's five Steps

6.1 Step 3 : Positioning

The third part of the docking algorithm is the Positioning. When the Robot is somewhere in the approach zone, it has to position in front of the Apriltag in order to dock. The robot can only make the last step of the docking algorithm when he stands right in front at the Apriltag. As in fig. 18 the robot stands in Point A and want to drive to Point P.

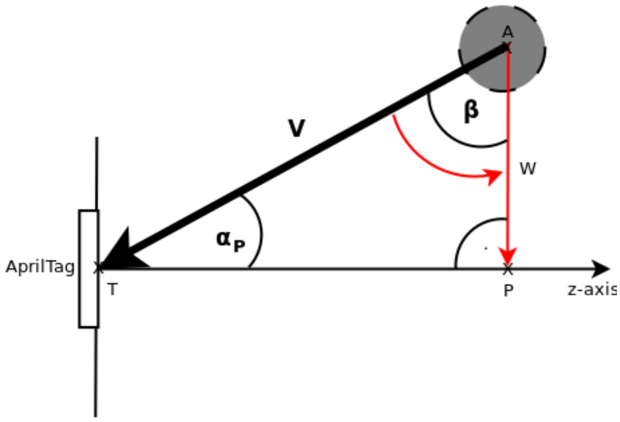


Figure 18: This figure shows the situation where the robot is after he arrived near the Apriltag by using the approach described in Phase 1. At Point A the robot still have to do the positioning in order to reduce the angle α.

Thus the angle α_P is to be measured which is the angle between $\vec{v}$ and z-axis. Once we have this angle we can compute the angle β which is the angle the robot has to turn. If we assume a right triangle (A,T,P) we know that the sum of all angles is $\pi = \alpha + \beta + \frac{\pi}{2}$ and with this equation, it follows:

$$\beta = \frac{\pi}{2} - \alpha$$

After the robot turned β to the left/right we have to compute the way. Because we have a right triangle, we can say that $sin(\alpha) = \frac{way}{|\vec{v}|}$. Thus the way must be:

$$way = sin(\alpha) \cdot \sqrt{(v.x)^2 + (v.z)^2}$$

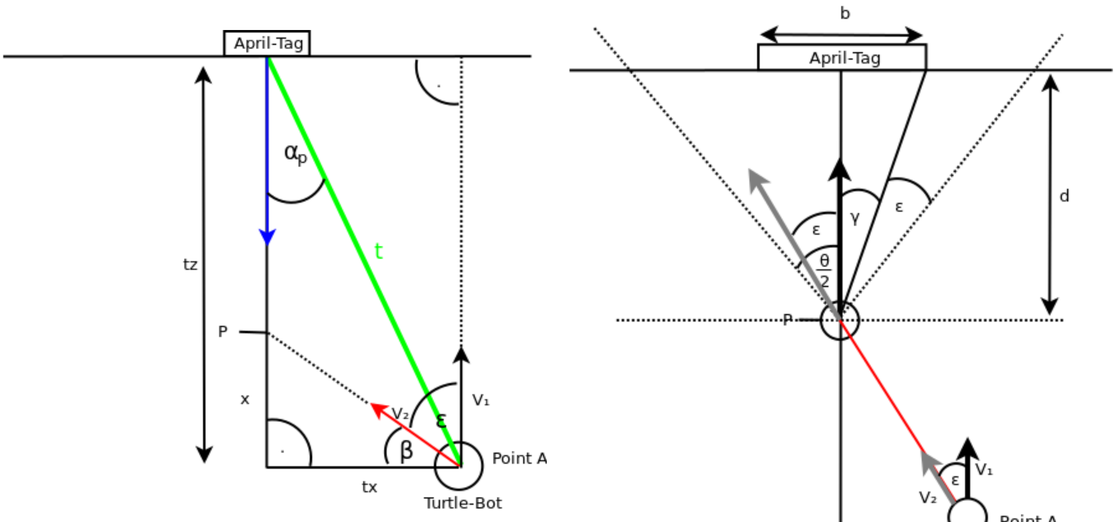
After the robot drove that distance he has to turn about $\frac{\pi}{2}$ right/left in order to look at the Apriltag. If the robot finished the positioning he decides if the next step is the linear approach or the frontal docking.

$$\alpha = \begin{cases} \text{linear approach} & \text{for } \alpha > 10.0^\circ \\ \text{frontal docking} & \text{for } \alpha \leq 10.0^\circ \end{cases} \tag{8}$$

If the angle is smaller than 10.0° the frontal docking algorithm can compensate that. Thus in this case we do not need the linear approach to the ideal path. But if the angle is bigger than 5.0° it is better to do the linear approach.

6.2 Step 4 : Linear approach

The fourth part of the docking algorithm is the linear approach to the ideal path. After the linear approach, the robot should stand with $\alpha_P = 0$ and α_{YAW} in front of the Apriltag. This approach is shown in fig. 19a. It is not possible to use dead reckoning by calculating the way the robot has to drive and use the function drive distance, because of the inaccuracies, which are about 3cm. Thus the robot has to see the Apriltag during the approach to the ideal path. As mentioned



(a) (b)

Figure 19: a) shows the situation before we begin with the fine positioning. b) shows the the situation when the robot is at point P.

before the Apriltag must be visible in order to detect if the robot has reached the ideal path. Thus in the Point P, the Apriltag must still be visible. This is the reason why the ε the robot has to turn in order to drive to the ideal path is not allowed to be too big. This angle must be as big as possible but small enough to keep the Apriltag in the camera field. If the robot loses the visual contact to the Apriltag, he cannot compute the angle α and does not know when to stop.
As shown in fig. 19b we can see that $\frac{\theta}{2} = \gamma + \varepsilon$ furthermore we can see that $tan(\gamma) = \frac{b}{2 \cdot d}$. Thus we can conclude the following equation:

$$\varepsilon = \frac{\theta}{2} - atan(\frac{b}{2 \cdot d}) \tag{9}$$

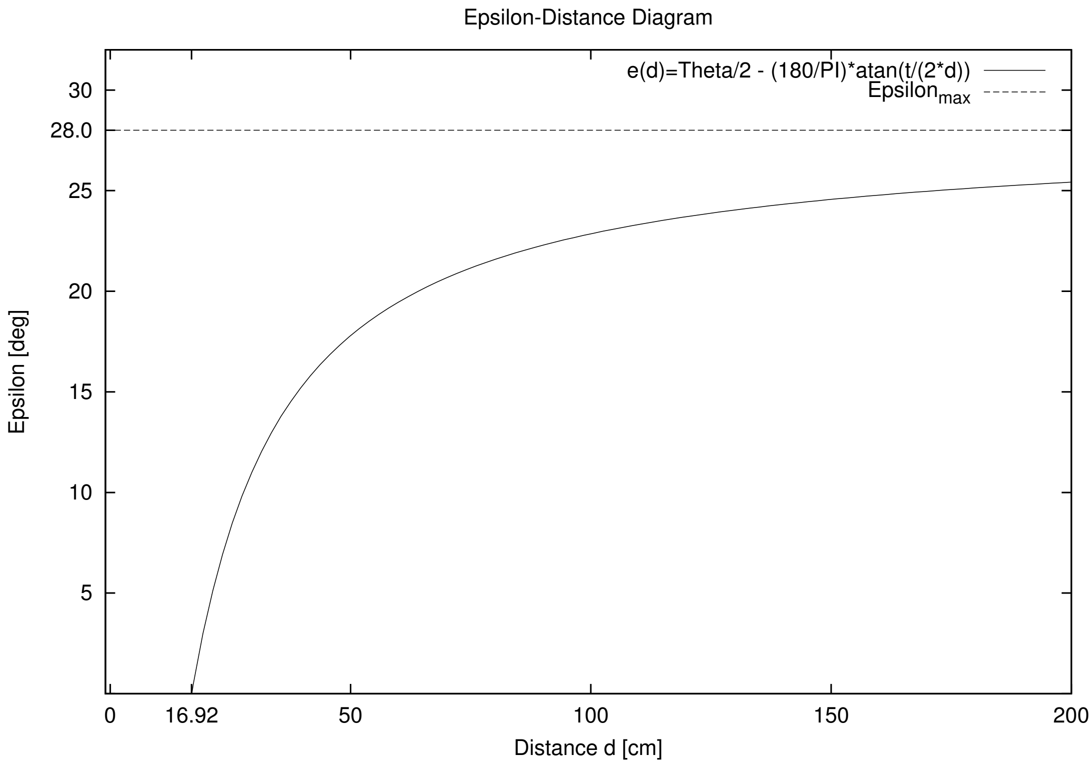


Figure 20: This Diagram shows the equation 9 with the parameters θ = 56° and b = 18cm

The angle ε is the maximal angle the robot can turn without losing the Apriltag out of the field of view. At the Point A we do not know the distance d, but we know that the Apriltag must be visible. Thus d_{min} is the distance which belongs to $\varepsilon(d_{min}) = 0$.

$$d_{min} < d < t_z$$

The angle ε_{max} with d $\in$ [d_{min}-t_z]. As shown in fig. **??** there are three more equations:
I : $\beta = \frac{\pi}{2} - \varepsilon$
II : $tan(\beta) = \frac{x}{t_x}$
The part x of the ideal path should be about $p = 25$ % of the entire ideal path $(x + d)$.
III: $p = \frac{x}{x+d} \Rightarrow x = \frac{-pd}{(p-1)}$ with $x = tan(\beta) \cdot t_x$

$$d = \frac{p-1}{-p} \cdot tan(\beta) \cdot t_x \tag{10}$$

Thus we have two equations:
I : $d = \frac{1-p}{p} \cdot tan(\frac{\pi}{2} - \varepsilon) \cdot t_x$

II: $\varepsilon = \frac{\theta}{2} - atan(\frac{b}{2 \cdot d})$
We calculate ε by subbing equation I into equation II then we solve this equation to ε. The complete derivation of the following equation is in the attachment at 11.1.
Finally, we get the following equations:
$P(p,t_x) := \frac{(b+a+A(p,t_x) \cdot a - b \cdot A(p,t_x))}{(1+ab \cdot A(p,t_x))}$ and $Q(p,t_x) := \frac{-(ab+A(p,t_x))}{(1+ab \cdot A(p,t_x))}$ and $A(p,t_x) := \frac{-b \cdot p}{2 \cdot (p-1) \cdot t_x}$

$$\varepsilon_{1,2}(p,t_x) = atan\left(\left(\frac{-P(p,t_x)}{2}\right) \pm \sqrt{\left(\frac{-P(p,t_x)}{2}\right)^2 - Q(p,t_x)}\right) \tag{11}$$

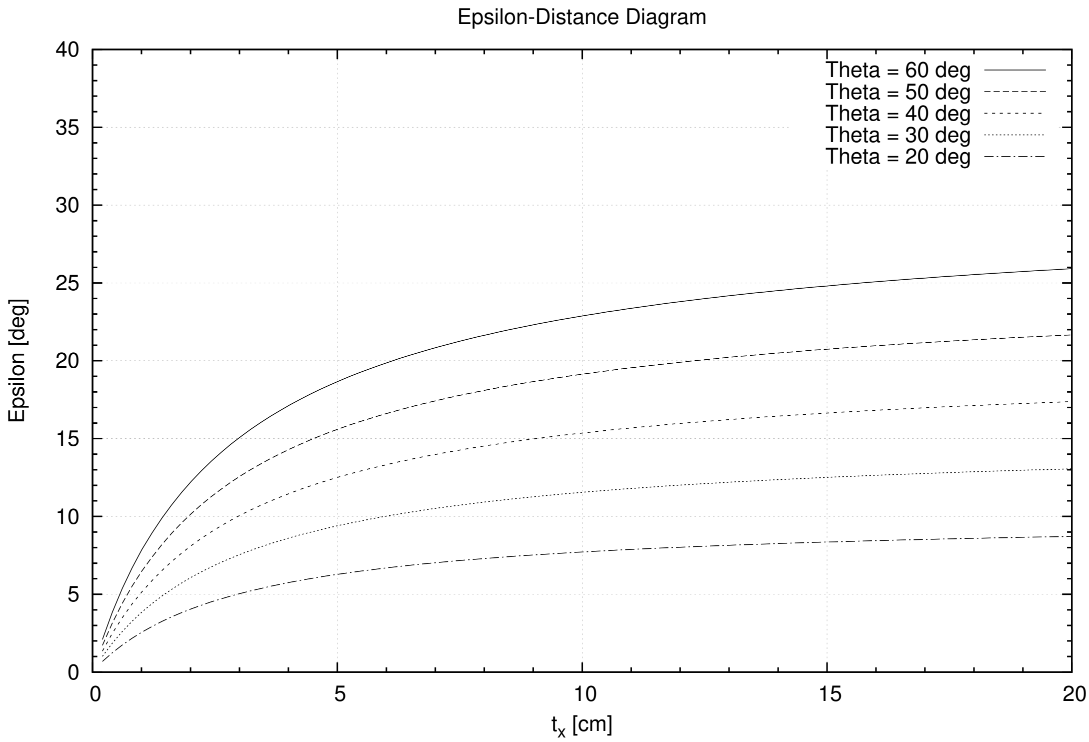


Figure 21: This Diagram shows the equation 11 over the interval of $t_x \in$ [0.0cm - 20.0cm]. The function is plotted for multiple parameters for θ as shown in the legend. The parameter p is set to 25%.

In theory, the linear approach works well, just the angle ε has to be computed and then the robot can turn ε deg, drive forward until the angle α_P becomes zero. Once α_P is zero you can turn ε deg in the opposite direction. And then the robot should stand in front of the Apriltag, with an angle α_P of nearly zero. One of the fundamental problems of this approach is the fluctuation of the measured angle α_P. Thus we use an average value of N angles to flatten these fluctuations. As describes in section 5.1 the function `get`$_{a}vg_{p}osition_{a}ngleisusedtocalculatetheangleinordertoavoidtheaveragingover$ `even during a drive.`

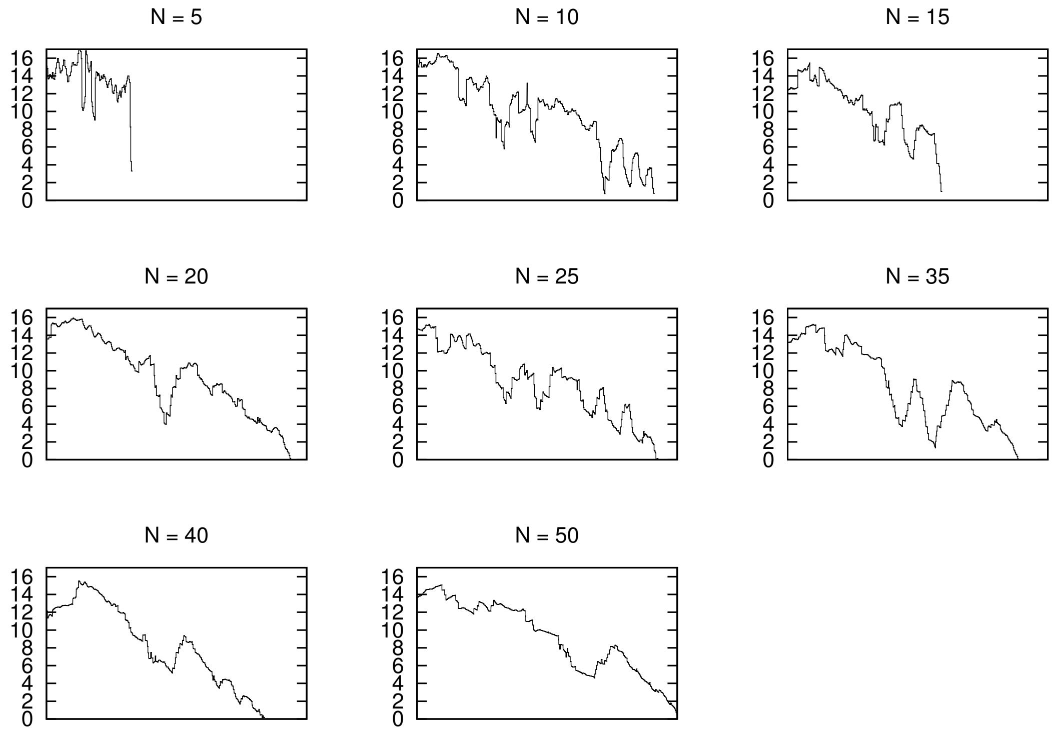


Figure 22: This figure shows the angle α_P over the iterations of the get_avg function. The measurement was taken during the linear approach (section 6.2) when the robot drives to Point P. The diagrams shows the angle α_P with a mean of N measured values during the drive at a velocity of $7\frac{cm}{s}$

`As shown in fig. 22 a reasonable value for N is 40, because the graph looks almost linear.`

6.3 Step 5 : Frontal docking

`The last part is the part where the robot sees the Apriltag and drives towards it,with small corrections of the angle` α`. In this Phase, no big changes of the position are performed.`
`As shown in fig. 23 the robot does some corrections if the position angle` α_P `is bigger than a certain threshold` α_T`. Thus we can write:`

Algorithm 2 Linear Approach

procedure LINEARAPPROACH
  Vector2 pos $\leftarrow$ getposition() $\triangleright$ Gives us pos.x=t_x and pos.y=t_z
  $\varepsilon \leftarrow \frac{pos.x}{|pos.x|} \cdot$ epsilon(|pos.x|) $\triangleright$ If the robot stands right ε is negative
  $\beta \leftarrow \frac{\pi}{2} - \varepsilon$
  d $\leftarrow pos.y - |pos.x| \cdot tan(|\beta|)$
  if $d <$ minDistance **then**
    moveAngle (π)
    driveForward(0.5)
    moveAngle $(-\pi)$
    linearApproach()
    break
  else
    moveAngle (ε) $\triangleright$ Turns right if $\varepsilon < 0$ and left if $\varepsilon > 0$
    $\alpha_{neu} \leftarrow 2 \cdot \pi$
    E $\leftarrow 0.3$
    while $\alpha_{neu} >$ E **do**
      $\alpha_{neu} \leftarrow$ | _avgPositionAngle |
      driveforward()
    moveAngle $(-\varepsilon)$

$$-\alpha_T < \alpha_P < \alpha_T \tag{12}$$

If the robot is in a position where $\alpha_T < \alpha_P$ holds, then the robot has to turn LEFT and has to drive forward until equation 12 holds again or α_P becomes zero. After the robot as driven this correction way he finally has to turn in the opposite direction. If the robot is somewhere where $-\alpha_T > \alpha_P$ we can do the same analog with a turn to the RIGHT. Now we still have to think about a suitable threshold angle α_T.

In order to find a suitable threshold value some measurements have been made where the robot use the frontal docking algorithm described in algorithm 3 and test multiple values for α_T.

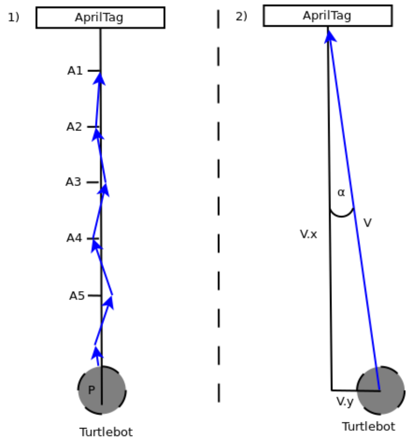


Figure 23: The figure shows the last part of the docking procedure. The blue vectors represent small correction the robot does during docking.

α_T[deg]	N	success [%]
0.5	13	0.23
1.0	27	0.37
1.5	30	0.44
2.0	31	0.53
2.5	17	0.41
3.0	15	0.33
4.0	15	0.27

Table 3: The measurements which have been done in order to find a suitable threshold value for α_T. For every α_T the success rate of successful docks is computed.

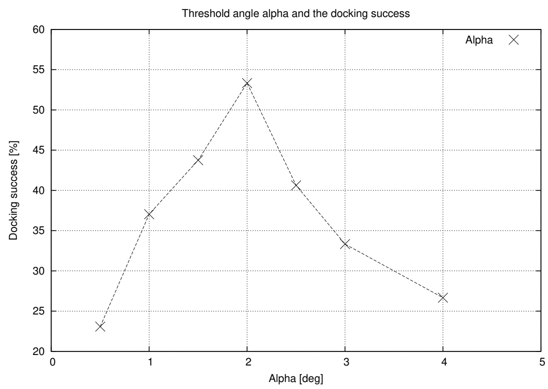


Figure 24: This is the diagram which belongs the measured values in table 3. The graph does have a peak at $\alpha_T = 2.0°$, which is the suitable threshold for α_T

The robot is put 68cm in front of the Apriltag with an position angle of 0° and $\varepsilon = 0°$, then the robot has to perform frontal docking. The measured value is the success rate. The result of these measurements is shown in table 3. If the threshold angle is too small the robot does stop to often in order to correct the angle and if the angle is too big the robot does not correct the course. Thus there are two things that can prevent a successful docking. The first are too many stops especially when the robot try's to drive in the docking station, the second are not enough corrections. As shown in table 3 an angle of 2.0° is the reasonable threshold.

Algorithm 3 Frontal Docking

procedure MOVE(ω, v) ▷ This function send a Twist-Message

procedure FRONTALDOCKING

Vector2 pos = getPosition()

$\alpha_T \leftarrow 2^\circ \cdot \frac{\pi}{180^\circ}$ ▷ This is the boundary angle

$\alpha_{dock} \leftarrow$ _avgDockingAngle ▷ We take 10 values for the mean

if $\alpha_{dock} > \alpha_T$ **then** ▷ We have to turn LEFT

$\omega \leftarrow 2^\circ \cdot \frac{\pi}{180^\circ}$

else if $\alpha_{dock} < -\alpha_T$ **then** ▷ We have to turn RIGHT

$\omega \leftarrow -1 \cdot 2^\circ \cdot \frac{\pi}{180^\circ}$

else

$\omega \leftarrow 0.0$

$v = 0.07$ ▷ This is the driven velocity

if pos.y $>$ STOP_DISTANCE **then** ▷ reached the end

move(ω, v)

frontalDocking()

else ▷ the robot stops driving forward

adjust() ▷ The robot turns until yaw angle is zero

sleep(3s)

if ! is_recharging **then**

drive forward(-0.65)

frontalDocking()

break ▷ Stop vehicle

7 Evaluation of the docking algorithm

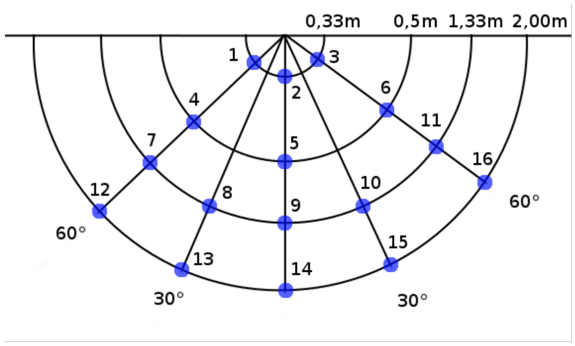


Figure 25: Testfield with startingpoints for evaluation

To evaluate the docking algorithm with others, we use the same approach as [9]. We use a test field which lies in the approach zone. Then we use 16 locations that are distributed over the test field. This field extends to 2m and 60° in both directions from the center, as shown in fig. 25. Furthermore, we set the zero point to the point where the `base_link` of the robot stands when is is docked. Because the left side of the test field is exactly the same as the right side, only the right side is part of the Evaluation. Thus we can summarize some points to reduce the number of measurements. For example, 8,10 becomes one Point, 13,15 and so on. In order to compare the new docking algorithm with the old one mentioned in section 3.1, measurements for both algorithms are made. Before a measurement is counted as a failure the robot is allowed to do the retry at the and of the frontal docking five times. Because the old approach from Kobuki does not have any retry functionality, only the new approach is influenced by this rule. As can be seen in table 4, the new docking approach works quite well. But at the Points 15 and 16 the robot was not able to dock correctly. This failures come from inexact detections of the Apriltag. If the Apriltag is constructed bigger e.g. 16cm² it should be possible to dock correctly. The range of the Apriltag gets slightly smaller, when looked from an angle on it. Because the tag was designed with 12cm², which is the size that correspondent to the maximal distance of 2m, the small reduced range at Point 15 and 16 causes some docking failures.
Still now the bumper on the robot is still activated during docking. If the robot drives into the docking station the bumper can be triggered, because the robot drives $\tilde{1}$cm to far. This is mainly the reason why we have the failed docking in Point 3.
The Kobuki's Docking Approach worked well in the Points 2,5,9 and 14. But it was not able to dock on all the other Points. One reason is that the robot tried to dock on a different docking station that was more than 5m away. The other reason is that the robot was not able to drive exactly in the docking station.

degree	Points	New Docking			Kobuki's Docking		
		sucess	missed	%	sucess	missed	%
0	2	6	0	100	6	0	100
0	5	6	0	100	6	0	100
0	9	6	0	100	6	0	100
0	14	6	0	100	6	0	100
30	8,10	7	0	100	0	6	0
30	13,15	8	7	50	0	6	0
60	1,3	8	1	87,5	0	6	0
60	4,6	6	0	100	0	6	0
60	7,11	6	0	100	0	6	0
60	12,16	8	10	44	0	6	0
all	2,3,5,6,9,10,11	45	1	97.8	18	24	42.8
all	all	67	18	78.8	24	36	40.0

Table 4: This Table shows the measurements that are taken to evaluate the new docking algorithm and compare it to the Kobuki's docking algorithm

8 Conclusion

It could be demonstrated, that the localization with Apriltags is precise enough to do the docking with a success rate of 100%. Because of the retry at the end of the new docking approach, the new approach does have a 38% higher success rate at a range of 2m. If we reduce the range to 1.33m we reach an improvement of 55 %. Furthermore, it is now possible to distinguish between multiple docking stations by using a different Apriltags. This was one of the main weaknesses of the old approach from Kobuki, which sometimes mixed up the docking stations if the infrared signal from two stations arrived at the robot. The major disadvantage of the new docking approach is the `drive_forward` function, which uses a calibration in order to drive an exactly a given distance. The first disadvantage is that a fixed acceleration must be used, the second, that the calibration only works on an even floor. If the floor is e.g covered with tiles, the calibration does not work properly anymore. If would be much better, when the `drive_forward` function would be rewritten, using sensors to measure the driven path, by counting the wheel revolutions or using the accelerometer in the IMU. Still now in both approaches, there is no collision control for obstacles during docking. Thus the approach zone should always be free of permanent and non-permanent obstacles. Furthermore, the used Apriltag must be visible for the robot's camera when the robot starts the frontal docking or the linear approach. This makes avoiding obstacles during docking very difficult, because if the obstacle covers the Apriltag, no optical based docking will be possible. One of the improvements that could be done by using the laser sensor is to detect non-permanent obstacles and then stop the docking in order to resume when the obstacle has gone. Especially interesting is the RFID-approach described in section 4.2. With this approach, there is no need to keep the Apriltag in the reach of vision. This could make it more easy to avoid obstacles during docking.

9 Bibliography

[1] "Documentation of Kinect." https://msdn.microsoft.com/en-us/library/jj131033.aspx. Accessed: 2017-03-27.

[2] "The Laser Sensor URG-04LX." http://docs.roboparts.de/URG-04LX_UG01/Hokuyo-URG-04LX_UG01_spec.pdf. Accessed: 2017-03-28.

[3] "PIXHAWK Internal Measurement Unit." https://pixhawk.ethz.ch/electronics/imu. Accessed: 2017-03-20.

[4] "The Kobuki TurtleBot." http://www.turtlebot.com. Accessed: 2017-03-19.

[5] "Testing Automatic Docking." http://wiki.ros.org/kobuki/Tutorials/TestingAutomaticDocking. Accessed: 2017-03-19.

[6] J. Borenstein and L. Feng, "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 869–880, Dec 1996.

[7] J. Borenstein and L. Feng, "UMBmark: A Method for Measuring, Comparing, and Correcting Dead-reckoning Errors in Mobile Robots," 1994.

[8] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–652–I–659 Vol.1, June 2004.

[9] B. W. Minten, R. R. Murphy, J. Hyams, and M. Micire, "Low-Order-Complexity Vision-based Docking," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 922–930, Dec 2001.

[10] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil, "LANDMARC: Indoor Location Sensing Using Active RFID," *Wireless networks*, vol. 10, no. 6, pp. 701–710, 2004.

[11] N. Kirchner and T. Furukawa, "Infrared Localisation for Indoor UAVs," 2005.

[12] M. Kim, H. W. Kim, and N. Y. Chong, "Automated Robot Docking Using Direction Sensing RFID," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 4588–4593, April 2007.

[13] "Sourcecode of the Apriltag-Package." https://april.eecs.umich.edu/software/apriltag.html. Accessed: 2017-04-01.

10 Attachment

10.1 Derivation of Equation 11

We have two equations:
I : $d = \frac{1-p}{p} \cdot tan(\frac{\pi}{2} - \varepsilon) \cdot t_x$
II: $\varepsilon = \frac{\theta}{2} - atan(\frac{b}{2 \cdot d})$
We calculate ε by plug in d from I in the equation II:

$$\Rightarrow \varepsilon = \frac{\theta}{2} - atan(\frac{b}{2 \cdot \frac{1-p}{p} \cdot tan(\frac{\pi}{2} - \varepsilon) \cdot t_x}) = \frac{\theta}{2} - atan(\underbrace{\frac{-b \cdot p}{2 \cdot (p-1) \cdot t_x}}_{=:A} \cdot \frac{1}{tan(\frac{\pi}{2} - \varepsilon)}) = \frac{\theta}{2} - atan(\frac{A}{tan(\frac{\pi}{2} - \varepsilon)})$$

$$\Rightarrow tan(\varepsilon) = tan(\frac{\theta}{2} - atan(\frac{A}{tan(\frac{\pi}{2} - \varepsilon)})) \underbrace{=}_{tan(A-B) = \frac{tan(A)+tan(B)}{1-tan(A) \cdot tan(B)}} \frac{tan(\frac{\theta}{2}) + tan(atan(\frac{A}{tan(\beta)}))}{1 - tan(\frac{\theta}{2}) \cdot tan(atan(\frac{A}{tan(\beta)}))} = \frac{tan(\frac{\theta}{2}) + \frac{A}{tan(\beta)}}{1 - tan(\frac{\theta}{2} \cdot \frac{A}{tan(\beta)})}$$

$$\Leftrightarrow tan(\varepsilon)(tan(\beta) - tan(\frac{\theta}{2}) \cdot A) = tan(\beta) \cdot tan(\frac{\theta}{2}) + A$$

$$tan(\varepsilon) \cdot tan(\beta) - tan(\varepsilon) \cdot tan(\frac{\theta}{2}) \cdot A - tan(\beta) \cdot tan(\frac{\theta}{2}) = A$$

To simplify the expressions we substitute:

$$x := tan(\varepsilon),\ a := tan(\frac{\pi}{2}),\ b := tan(\frac{\theta}{2})$$

and with $\beta = \frac{\pi}{2} - \varepsilon$ we can write $tan(\beta)$ as:

$$tan(\beta) = tan(\frac{\pi}{2} - \varepsilon) \underbrace{=}_{tan(A-B) = \frac{tan(A)+tan(B)}{1-tan(A) \cdot tan(B)}} \frac{tan(\frac{\pi}{2}) + tan(\varepsilon)}{(1 - tan(\frac{\pi}{2}) \cdot tan(\varepsilon))} \underbrace{=}_{substitution} = \frac{a+x}{(1-a \cdot x)}$$

using this substitutions gives:

$$x \cdot \frac{a+x}{1-a \cdot x} - x \cdot b \cdot A - \frac{a+x}{1-a \cdot x} \cdot b = A$$

$$\Leftrightarrow \frac{x(a+x)}{1-a \cdot x} - \frac{x \cdot b \cdot A \cdot (1-a \cdot x)}{1-a \cdot x} - \frac{a+x}{1-a \cdot x} \cdot b = A$$

$$\Leftrightarrow x \cdot (a+x) - (x \cdot b \cdot A \cdot (1-a \cdot x)) - b \cdot (a+x) = A \cdot (1-a \cdot x)$$

$$\Leftrightarrow ax + x^2 - x \cdot b \cdot A + b \cdot A \cdot a \cdot x^2 - ab + bx - A + Aax = 0$$

$$\Leftrightarrow x^2 + b \cdot A \cdot a \cdot x^2 + bx + ax - A + Aax - x \cdot b \cdot A - ab = 0$$

$$\Leftrightarrow (1 + abA) \cdot x^2 + (b + a + Aa - bA) \cdot x - (ab + A) = 0$$

$$\Leftrightarrow x^2 + \frac{(b + a + Aa - bA)}{(1 + abA)} \cdot x + \frac{-(ab + A)}{(1 + abA)} = 0$$

now we can substitute again with:
$P(p,t_x) := \frac{(b+a+A(p,t_x)\cdot a-b\cdot A(p,t_x))}{(1+ab\cdot A(p,t_x))}$ and $Q(p,t_x) := \frac{-(ab+A(p,t_x))}{(1+ab\cdot A(p,t_x))}$ and $A(p,t_x) := \frac{-b\cdot p}{2\cdot(p-1)\cdot t_x}$

$$x_{1,2}(p) = \left(\frac{-P(p,t_x)}{2}\right) \pm \sqrt{\left(\frac{-P(p,t_x)}{2}\right)^2 - Q(p,t_x)}$$

$$\varepsilon_{1,2}(p,t_x) = atan\left(\left(\frac{-P(p,t_x)}{2}\right) \pm \sqrt{\left(\frac{-P(p,t_x)}{2}\right)^2 - Q(p,t_x)}\right) \tag{13}$$

10.2 Code of the 3DModel

```
//  Written by Kolja Poreski
//<9poreski@informatik.uni-hamburg.de>,<Poreski@gmx.de>
//
translate([0,0,15]){
     difference(){
     cube([8,2,8],center=true);
          translate([0,0,-4]){
               cube([5.1,1,5],center=true);
          }
     }
}
translate([0,0,7]){
     cube([5,0.97,4],center=true);
}
translate([0,0,0]){
     cube([1.5,0.97,11],center=true);
}
translate([0,0,-6]){
     difference(){
     cube([5,0.97,2],center=true);
     for(i=[-1,1]) {
               translate([i*1.5, 0, 0]) {
                    union(){
                         cylinder(r=0.23, h=100, center=true , $fn=60);
                          for(i=[0:3]) {
                              translate([0,i*0.3,0])
                              rotate([0,0,30])
                             cylinder(r=0.43, h=0.4, center=false , $fn=6);
                         }
                    }
                    }
               }
     }
}
translate([0,0,-10]){
     difference(){
          union(){
               cube([10,4,1],center=true);
               translate([0,0,1]){
                    cube([6,2,2],center=true);
               }
          }
          translate([0,0,1.5]){
               cube([5.1,1,2],center=true);
          }
          for(i=[-1,1]) {
               translate([i*1.5, 0, -0.52]) {
                    cylinder(r=0.23, h=100, center=true , $fn=50);
                    cylinder(r=0.5, h=0.4, center=false , $fn=50);
               }
         }}}
```

10.3 Linear Regression Method

P is a given set of two dimensional Points.
For example: $P = \{(1.0, 2.0); (3.0, 2.0); (3.0, 1.7)\}$
The calculation of a regression line that minimizes the mean square error can be done with the following equations:

$$m = \frac{\sum_{p \in P} x \cdot y}{\sum_{p \in P} x^2} \text{ and } b = \frac{1}{|P|} \left(\sum_{p \in P} (y) - m \cdot \sum_{p \in P} (x) \right)$$

$$f(x) = m \cdot x + b$$

10.4 Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Wirtschaftsinformatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den April 5, 2017 Vorname Nachname

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den April 5, 2017 Vorname Nachname