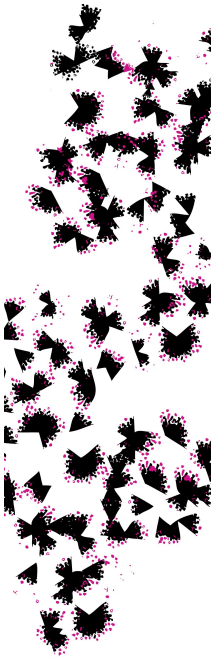


Vision-based control of the SHERPA arm

R. (René) Meijering

MSc Report

Committee:
Dr.ir. J.F. Broenink
Dr.ir. R.G.K.M. Aarts
Dr.ir. J.B.C. Engelen
M. Reiling, MSc

January 2018

002RAM2018
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The SHERPA project focusses on the smart collaboration between humans, ground robots and aerial robots, in improving rescue activities in alpine environments. Aerial support, among others, is carried out by small-scale Unmanned Aerial Vehicles (UAVs). A downside of using UAVs, is the relative short time of flight due to limited battery capacity. In this context an autonomous way of retrieving the UAVs is required, for the purpose of battery exchange. A mobile ground station, equipped with a robotic arm, is conceived for this task.

The aim of this thesis is to design and implement the autonomous grasping, docking and deployment procedure of a small-scale UAV by the mobile ground station. The UAV has to be docked on a battery exchange station, mounted on this platform. Two scenarios are considered for the execution of this task. In the first scenario a UAV has safely landed in the proximity of the mobile ground station. The UAV is localized by the mobile ground station and retrieved for battery exchange. The second scenario involves grasping a hovering drone in case it is not able to land safely. For this purpose the robotic arm has to be able to accurately follow the movements of the hovering drone.

In this work, a vision-based-marker-system is implemented on the existing robotic platform. A monocular camera, mounted on the robotic arm, is used for localization and pose estimation of the UAV. By updating the current software architecture of the task planner a flexible platform is provided. With the new architecture, the task described in scenario one was successfully implemented. Unfortunately the platform is not well suited for the task described in scenario two due to high latencies and the platforms control architecture. To overcome the effects of the latencies, two trajectory prediction methods are implemented. From the test result can be concluded that the prediction methods helps in obtaining a more accurate 'real time' position of the UAV. However, these results were not successful enough to prove this system can be used for grasping a hovering drone.

In continuation of this project, it is recommended to extend the amount of recovery procedures to increase robustness. More feedback information should be send towards the SHERPA delegation framework about its current status. In case the UAV is not positioned correctly for battery exchange a re-position procedure should be performed. In order to track a hovering drone using the robotic arm it is advised to implement a high speed, global shutter camera, for better vision performance. Analyse the velocity and acceleration limits of the arm to allow faster motions or implement velocity control. In order to reduce the delay in the system a different velocity filter is advised in combination with a variable prediction time step to overcome varying processing delays.

Contents

1 Introduction

1.1 Context

1.1.1 SHERPA project

Introducing robotic platforms in rescue operations offer a promising solution for the improvement of search and rescue activities. Such rescue activities are typically characterized by large areas of territory, adverse terrain and changing weather conditions. In these conditions the area must be patrolled efficiently, while keeping the risks for human beings at reasonable levels. For example, during a rescue operation in the alpine scenario.

Within this context a project named "Smart collaboration between Humans and ground aErial Robots for imProving rescuing activities in Alpine environments" (SHERPA) was launched (Marconi et al., 2012). The goal of SHERPA is to develop a robotic platform for supporting the rescuers in their activity and improving their ability to intervene promptly. The activities of SHERPA focusses on a rescue team with "Smart collaboration between humans and ground-aerial robots". Within the scope of this research, the small-scale Unmanned Aerial Vehicles (UAVs) and ground rover are of particular interest:

Small-scale UAVs
Small scale rotary-wing UAVs are used to support the rescuing mission by enlarging the patrolled area with respect to the area potentially 'covered' by the human rescuer. The small-scale UAVs, also called 'trained wasps', can be equipped with cameras and other sensors / receivers. These sensors are used to gather visual information and the monitor emergency signals. As a consequence, their payload, operative radius and time of flight are limited by the battery capacity.

Ground rover
The Ground rover, also called 'Intelligent donkey', acts as a mobile ground station. It contains a hardware station with computational and communication capabilities and a battery exchange station for the small-scale UAVs. The battery exchange station is a separate module, also referred to as the 'SHERPA-box'. The ground rover is technically conceived to operate with a high-degree of autonomy and long endurance. To improve its autonomous capabilities, a multi-functional robotic arm is installed. This arm will be used for docking and deployment of the small scale UAVs on the SHERPA-box.

1.2 Problem statement

In SHERPA, it is envisioned to exchanging the UAV its battery directly in the field, to prolong their mission time. The task of exchange the batteries must be autonomously executed by the SHERPA ground rover.

At the starting-point of this research, the SHERPA project has been under development for approximately forty-two months of its forty-eight months duration. Multiple systems, like the ground rover and robotic arm, have already been developed and implemented. Other systems, like the small-scale UAVs, are still under development. However, the task of autonomous battery exchange of the UAVs is yet to be implemented, and will be the aim of this research.

For the task of autonomous battery exchange the following two scenarios are considered:

Scenario 1: Docking a landed drone
The UAV has safely landed in the proximity of the ground rover. The exact location of the UAV needs to be determined. If the UAV is located within the workspace of the robotic arm, the arm should reach, grasp and dock the UAV on to the SHERPA-box. After battery exchange the drone is deployed back in the field, ready for take-off.

Scenario 2: Grasping a hovering drone
In case the UAV is not able to land safely, an in-flight grasping procedure is envisioned. The robotic arm will attempt to grasp the UAV while hovering. For this purpose the system needs to be able to follow the movement of a hovering drone.

In both scenarios the GPS signals of the ground rover and UAV can be used to bring both vehicles in close proximity of each other. However, the accuracy of this proximity could be up to a couple of meters, due to inaccurate GPS localization. It is assumed that the rover is able to autonomously drive to the location of the UAV. From this point a vision-based system is envisioned to locate the actual position of the UAV. This vision system is not yet present and has to be implemented in the current architecture. The vision-based information will be used to control the robotic arm. To do this the software architecture has to be modified. These changes, with respect to the current platform, are depicted in Figure 1.1. The software for controlling the robot arm is running on a dedicated computer and is mounted on the ground rover.

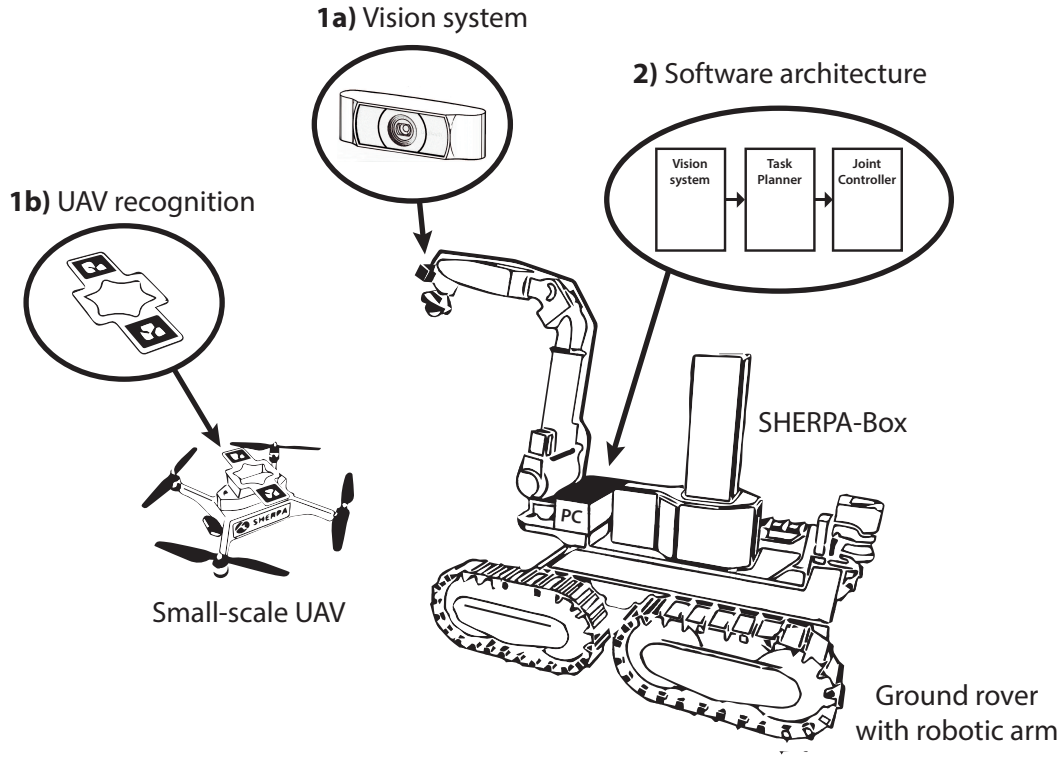


Figure 1.1: Overview of the elements that will be implemented or updated with respect to the current system

The SHERPA-Box, depicted in Figure 1.1, is a separate module. The actual battery exchange is part of this module and developed separately by another project partner. This research will therefore only focus on the design and implementation of the autonomous functionality, like

docking the small-scale UAVs on the SHERPA-Box. More information on the SHERPA platform is presented in Appendix A.

To summarize, the goal of this research is to design and implement the following elements in the current platform:

- Implementation of a vision-based system.
- Update the current software architecture.
- Autonomously grasp, dock and deploy a landed UAV with the SHERPA arm, using vision-based information
- Tracking the movement of a hovering UAV with the SHERPA arm.

1.3 Prior work

Prior work on several research topics has been performed within the Robotic and Mechatronics research group at the University of Twente. A custom robotic arm has been developed, containing two variable stiffness actuators (Barrett et al., 2017; Tan et al., 2017). To control the robotic arm an architecture is developed for communication between the local controller on the robotic arm and ROS (Boterenbrood, 2015). For grasping the small-scale UAVs a gripper is constructed, containing an interface mounted on the UAV (Werink, 2016; Barrett et al., 2016).

Motion control in Cartesian space allows collision-free path-planning of the robotic arm (Barbieri, 2016). This platform will act as the starting point for this research.

On the subject of visual tracking and grasping UAVs, prior work has been executed, using a camera in combination with the UAVs accelerometer data to obtain an accurate pose of the UAV (Baarsma, 2015).

1.4 Report Outline

In this report the vision system for UAV localization and pose estimation is presented in Chapter 2. The current control architecture is modified for the autonomous grasping, docking and deployment of the UAV in Chapter 3. Experiments performed with the modified system are presented in Chapter 4. With the main elements implemented, the tracking capability of the SHERPA arm is investigated in Chapter 5, followed by the conclusions and recommendations in Chapter 6.

2 Vision System

2.1 Vision analysis

From the scenarios described in the previous Chapter, the requirements for the vision system are derived. With these requirements a closer look is taken at the different types of identification systems, the one most suited with respect to the requirements is selected.

2.1.1 Requirements

From the scenarios in Section 1.2 the following requirements are derived:

Requirement 1: *The vision system must acquire a sufficiently accurate pose estimate*

In scenario 1 a landed UAV is retrieved within the workspace of the arm. The workspace of the arm is everything within a radius 1 m with respect the base of the robotic arm (Barrett et al., 2017). Within this workspace an accurate pose estimate is of importance during grasping. For a successfully grasping procedure the gripper must within a 3 cm tolerance from the centre of the gripper interface. Within this tolerance, the design of the gripper and its variable-stiffness joint allows for grasping a UAV.

Outside the arms workspace, a rough indication of the UAV pose is sufficient as the rover has to drive towards it. It is assumed that the rover is able to get within 2 to 3 meters from the landed UAV using GPS information. For localization the vision system should be able to determine the UAV its pose up to 3 meters. The allowable tolerance for localization is set to 40 cm.

Requirement 2: *The vision system must be implemented in the current operating platform*

The current hardware and software is all connected using the Robot Operating System (ROS) (Quigley et al., 2009). Within the SHERPA project this platform was chosen as the operating system. Therefore, the vision system should be able to connect with ROS. To do this a ROS-package, c-library, python script or other interface must be available for the vision system.

Requirement 3: *Each UAV should be uniquely identifiable*

In the SHERPA project multiple UAVs work together, to patrol a certain area. Therefore multiple UAVs might be present in a scene. Unique identifications allows the ground rover to retrieved the right one.

Requirement 4: *The vision system should use a passive technique*

The first scenario revolves around the limited battery capacity of the small scale UAVs. In this sense, a passive based vision system is highly favourable. Passive-based refers to a system that does not requires any kind of active signal from the target object used for localization. Active systems require power to operate, which is already a limiting factor in the stated situation. When the power of a battery is drawn to such extent, that an active system is not able to operate, the UAV cannot be found, which is undesirable.

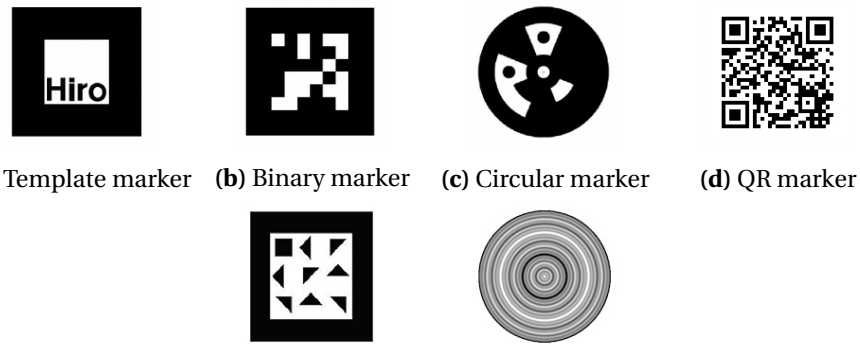
Requirement 5: *Any patterns or features should be scalable*

At this moment only a rough design of the UAV is available. Meaning detailed information like sensor placement, rotor size and usable surface area are unknown. In case, any predefined shape or pattern is required by the vision system, scalability and easy physical implementation is desirable.

2.2 Vision detection methods

Within computer vision different techniques exist that could be used for the recognition of objects. For example the object's shape could be used for identification. Packages like Object Recognition Kitchen (ORK) are available that can do this. This system uses a database of 3D models for the objects to recognize (Willow Garage, 2013). A disadvantage of such a system is that two objects of the same shape cannot be distinguished from each other. For example if two drones are of the same shape, both will be recognized as being a drone. However, they cannot be uniquely identified. Secondly the final design of the UAVs is not yet known. Therefore distinguishable features are also unknown, if any. For this reason, an object recognition technique is not very suitable.

Another technique is to recognize predefined patterns. These patterns are often referred to as *Fiducial markers* in literature, for readability the term *marker* is used instead. Such markers could be attached to any flat surface area and offer a certain amount of freedom in its physical application. When placed in a scene, a markers act as a reference frame, from which its relative pose can often be estimated. Markers come in a wide variety of shapes and patterns, several examples are presented in Figure 2.1.



(a) Template marker **(b)** Binary marker **(c)** Circular marker **(d)** QR marker

(e) Tricode marker **(f)** Fourier marker

Figure 2.1: Different marker designs, **(a)** Template marker (DAQRI, 2017), **(b)** Binary marker (Alvarn, 2016), **(c)** Circular marker (Siltanen, 2012), **(d)** QR marker (QR.Generator, 2014), **(e)** TriCode marker (Mooser et al., 2006), **(f)** Fourier marker (Sattar et al., 2007)

The design of these markers primarily depend on their field of application and the underlying detection algorithm. For instance, the QR marker depicted in Figure 2.1d, is designed to carry a high amount of data, While other markers are often used for identification and pose estimation. These techniques are often applied in Augmented Reality (AR) or tracking applications (Siltanen, 2012). Such systems are a promising solution for identifying the SHERPA drones.

For the purpose of marker identification and pose estimation, several marker designs are suited. To determine the most suited design, with respect to the requirements, several designs are compared.

2.2.1 Marker system comparison

In order to choose a suited marker system, different designs are compared. For this comparison the marker types mentioned in Figure 2.1 are used. Other marker designs exists, however, a full comparison goes beyond the scope of this research. The designs mentioned previously are the ones frequently found in literature. Most of these markers are designed to provide a pose estimate in combination with a unique identity or ID number. This ID number is often coupled to a database with specific information. This structure is also applicable for this project.

	Marker type					
Requirements	Template	Binary	Circular	QR	Tricode	Fourier
Acquires pose estimate	✓	✓	✓	✗	✓	✗
Multiple-marker-detection	✓	✓	✓	✗	-	-
Large distance detection*	~	✓	✓	✗	~	~
Ros \C++ \Python Library	✓	✓	✗	✓	✗	✗

* Distance up to 3 meters

✓ Implemented\found
~ Not very suited
- Unknown
✗ Not implemented\not found

Table 2.1: Marker design comparison

In Table 2.1, different marker designs and requirements are stated. The multi-marker detection requirement was previously not mentioned. Using a multi-marker setup helps in when a marker is occluded. In this situation other marker might still be visible, adding robustness to the system and therefore added. The comparison in Table 2.1 only considers black and white marker systems. The reasoning behind it, is that differences in brightness are easier to detect than differences in colour. This difference has often to do with the poor automatic white balance of cameras. Because changes in brightness are easier to detect, high contrast markers are favourable. In this case, black and white markers are optimal (Siltanen, 2012).

It can be noticed that the *Binary marker* scores positive on all requirements. Literature show that digital method based marker processing methods are most reliable. Their behaviour in the presence of errors can be predicted an corrected for (Kohler et al., 2010). Secondly, a binary marker, with a large physical cell size, is recommended for detection at relative large distances (Siltanen, 2012). In this context the large physical cell size refers to the internal binary grid size of the marker. The smaller the grid size, the larger the cells become, example are a 3x3 or 4x4 grid. This type of marker identification process is widely available through toolkits and libraries. *VTTs Alvar*, *April Tag* and *ARToolkit* are well-known examples of such toolkits (Alvarn, 2016; APRIL, 2016; DAQRI, 2017).

Conclusion

AR Track Alvar was chosen to be implemented on the SHERPA Platform. This library uses the VTT Alvar toolkit (Niekum, 2017). This package supports the use of binary markers, various grid sizes and contains useful features like bundle recognition. Bundle recognition allows for multiple markers to be recognized as on and eases on the multi-marker implementation. The comparison between different package is covered in Appendix B.

2.3 Camera system

For the marker detection system to work, an image capturing device or 'camera' is required. In this report, the vision camera used for marker detection is referred to as camera. This camera still has to be implemented on the platform. The placement of the camera might cause certain limitations or restrictions that have to be taken into account. Therefore the placement of the camera is determined first.

2.3.1 Camera placement

For the integration it was chosen to mount the camera on the gripper side of the robotic arm. This way, an entire area could be scanned, by moving the robotic arm. Another advantage is that continuous tracking is possible while grasping or driving around with the rover. A possible disadvantage is motion blur, due to the movement of the camera. Motion blur can occur when recording fast-moving objects, or in this case the camera that is moving. Motion blur results in distorted or incomplete captured images and could drastically decrease the performance of the vision system. The effects of motion blur depends on the type of image capturing sensor in the camera that is used. In general, a global shutter camera is most suited to record fast moving objects. In contrast to a rolling shutter camera, that often deals with motion blur (RED.COM, 2017).

A second limitation would be the allowable size and weight of the camera. The SHERPA-arm is a custom designed robotic manipulator with a load capacity of approximately 2 Kg (Barrett et al., 2017). Adding a relatively large or heavy camera will reduce its load capacity and motion freedom.

The main focus at this point in the project is on grasping and docking a landed UAV. In this scenario the effects due to motion blur can be minimized, by tuning the motion of the arm correctly. For instance, using relatively low speeds and accelerations. However, for tracking purposes, this becomes an issue. The camera will be placed near the gripper, therefore criteria like weight and size are of importance.

2.3.2 Camera selection

Within the scope of the project a camera was already purchased, namely the VI-sensor by Skybotix (Nikolic et al., 2014). However, this sensor was never tested for its intended use.

To confirm that the VI-sensor is suited for this application, it is compared with other cameras. Different camera were available within the research group for comparison and are listed below.

- Creative Live - Opita
- Logitech HD Pro C920
- Xbox 360 - Kinect
- Bumblebee 2

The cameras listed above differ in multiple aspects, ranging from a monocular-rolling-shutter cameras to stereo-vision-global shutter-cameras. Stereo vision cameras can be used to measure distances by performing triangular calculations (Manaf A. Mahammed, 2013). These kind of sensor are also often used in robotic applications in order to compute a point-cloud of a certain area (Patrick Mihelich, 2017).

The technical specification of these cameras, including the VI-Sensor, are presented in Appendix C.

From the technical specifications the *Xbox 360 - Kinect* could quickly be discarded. With a weight of approximately 1.36 kg, it would reduce the payload for grasping with 67.5%. This kind of capacity reduction is unacceptable. Secondly the Kinect sensor is less suited for out-

door applications. The sensor loses a part of its functionality in outdoor applications like these (Pagliari et al., 2016). The Bumblebee 2 camera turned out to be incomplete and could not be used.

Comparing the technical specification of the remaining sensors the *Logitech HD Pro C920* scores well on properties like weight, size and resolution. A disadvantage, is being a monocular-rolling-shutter camera. Making it less suited for distance calculation and tracking fast moving objects (RED.COM, 2017).

Obtaining sufficiently accurate pose information from the vision system is an important aspect. Performance wise a stereo camera, in combination with the marker detection system, would be ideal. This would combine depth information from the stereo camera with the flexibility of a marker system. The chosen *AR Track Alvar* package supports (besides monocular use) also depth information, however, is designed to be used with a Kinect camera. This feature was tested using depth information generated with the help of the ROS *Stereo Image Proc* package in combination with the VI-sensor(Patrick Mihelich, 2017). Unfortunately the test concluded unsuccessful, because the generated depth information was of the incorrect format for AR Track Alvar. Due to time constraints, no further research was conducted.

Conclusion

In this setup, any chosen stereo camera (besides the Kinect) would be used as a monocular camera, due to the software constrains. From this point of view the implementation of a stereo camera would be useless. For this reason and the properties of being a lightweight, small sized and high resolution camera, the *Logitech HD Pro C920* has been implemented.

2.4 Vision accuracy measurement

Several measurements were preformed to test the chosen vision system. During these measurements the focus was on discovering the detection range and accuracy. From the test results can be concluded if the system is accurate enough for the stated requirement. The following tests were preformed:

Distance & Accuracy
The goal of the first test is to determine the accuracy of the measured distance with the vision system. A 44x44 mm marker with a grid-size of 5x5 is placed along a measurement tape. The marker its distance, with respect to the camera is varied between measurements. In this test setup the measured and actual distance, of the marker are compared. This test was performed twice, using different camera resolution.

Marker angle & Detectability
A second test was preformed to determine the detectability of the marker, when placed under a predefined angle. The maximum detecting distance of this marker is measured. The goal of this measurement is to determine the minimum angle necessary to detect a marker from a certain distance.

The measurement setup and their corresponding test results are presented in Appendix D.

2.4.1 Results

From the measurement results in Appendix D can be concluded that a 44x44 mm marker can be detected up to 3 meters with a relatively small error, ranging from 0 to 11 cm (Table D.1). Within the workspace of the arm the error is even smaller, often a couple of mm. These results are well within the ranges specified in requirement 1. The distance test using the full resolution of the camera resulted in a more stable, and accurate pose estimated and is able to detect a marker up to a distance of 4.2 meters, versus 3 meter using the lower resolution. This outcome is not surprising as the full resolution measurement contains more pixels for detection and computation, allowing a more accurate calculation of the distance (Siltanen, 2012).

From the results in Table D.2 can be concluded that a marker angle of 30° or higher is sufficient for stable detection within the workspace of the arm. An increase in angle shows results in an increase in detection distance, meaning a larger angle is required for detecting a UAV outside the workspace of the arm.

Using these test results it was decided to use the low resolution setting, as it less computationally heavy and sufficiently accurate. If possible the markers will be placed at a relatively large angle with respect to the camera. In this case it is assumed that the camera is facing downwards when searching for the drone.

2.5 Vision implementation

This section covers the integration of both maker and camera-system. The chosen software package and its connection within the software architecture is explained. In the last section an overview of the implemented hardware components is presented.

2.5.1 ROS package

The chosen marker package has to be implemented within the overall architecture. To connect the package to other parts within this architecture three data types are published, namely:

Topics:
The *visualization marker* is a rviz message to display a squared block at the location of each identified markers.
The *ar pose marker* topic publishes a list of the poses of all the observed AR tags, with respect to the output frame.

Transforms:
The *tf_transform* provides a transform from the camera frame to each AR tag frame, named ar_marker_x, where x is the ID number of the tag.

The received marker pose is with respect to a certain output reference frame, in this case being the camera frame. This frame is added to the overall robot model, which is defined in the form of Unified Robot Description Format or (URDF) file. This allows the calculation of the marker pose with respect to any frame within the robot model, for example the gripper frame. Figure 2.2 presents the graphical representation of the robot model in combination with the marker detection system.

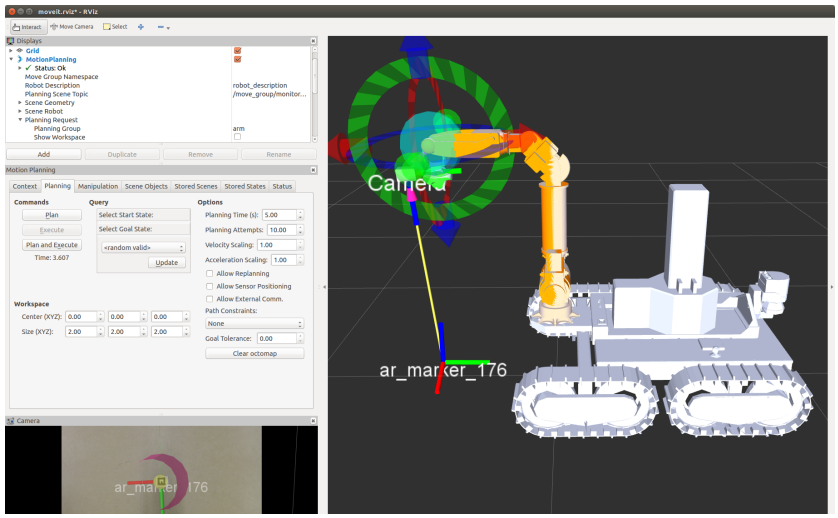


Figure 2.2: Marker detection in combination with the robot model

Knowing the marker position with respect to the gripper frame provides a set point for the end effector. This set-point enables the control of the SHERPA arm using vision based information.

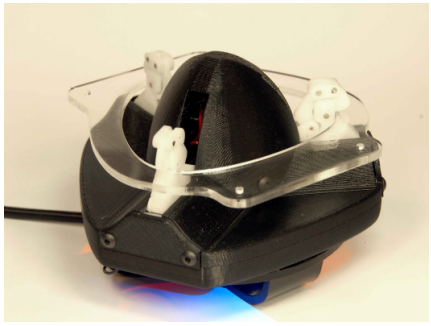
2.5.2 Hardware implementation

The integration of the vision system requires the implementation of certain hardware components like the camera and markers.

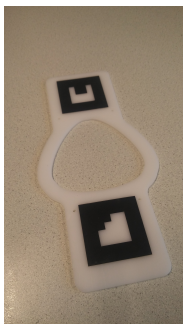
Marker placement
During the process of marker system selection and implementation the final design of the

small-scale UAVs was unknown. The only certainty was the attachment of an interface on the drone used for grasping. This interface is depicted in Figure 2.3a. It was decided that this interface will be redesigned to hold space for the markers, as shown in Figure 2.3b.

The markers are constructed of self adhesive vinyl with a size of 44x44 mm. Using a blackboard based vinyl ensures low light reflectance of the marker itself. Light reflection causes distortions during the marker detection process and is therefore of importance. For the same reason the gripper interface has been sanded to give it a matte finish.



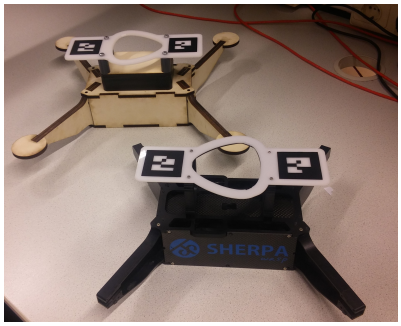
(a) Original gripper and gripper interface (Barrett et al., 2016)



(b) Updated gripper-interface

Figure 2.3: Original and modified gripper interface

The redesigned gripper interface was attached to the available mock-up drones and later on a finalized SHERPA drone as depicted in Figure 2.4.

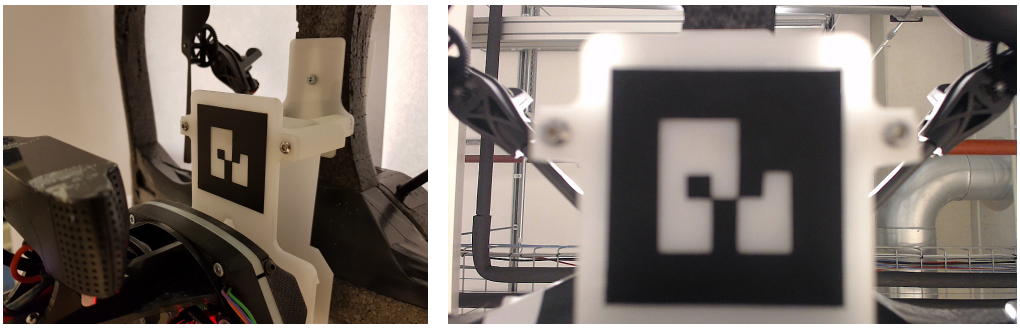


(a) Mock-up drones

(b) Actual Sherpa drone

Figure 2.4: Gripper interface mounted on the mock-up and actual SHERPA drone

Camera placement
In section 2.3.1 it was concluded that the camera will be placed near the gripper. During installation it was decided to place the camera just behind gripper, depicted in Figure 2.5. This placement allows a full field of view for the camera and almost no limitations towards the grippers freedom in motion. Secondly the marker can always stay in-line of sight while grasping. When grasped the marker is centred with the camera and still recognized by the marker software. In this configuration the vision information could be used throughout the procedures of grasping, docking and deployment.



(a) Marker in field of view of the camera **(b)** Screenshot of the camera view

Figure 2.5: The marker stays visible when grasped

2.5.3 Package limitations

During implementation and testing of the AR Track Alvar package several problems and limitations were encountered. One of them was the usage of the marker bundle feature, which resulted in an unstable orientation estimate once implemented on the (mock-up) drones. To resolve this issue the individual marker detection setting was used. Meaning all markers are recognized individually. By measuring the covariance of both marker orientations, the one with the lowest value and therefore most stable, is chosen as its reference marker.

Another limitation was the rather low output update frequency of 10 Hz using bundle recognition. This value was predefined within the software and could not be updated in the parameters. By switching to the individual marker detection setting the update frequency could be updated to 30 Hz. In the individual marker detection node the update frequency could be adjusted in the parameter settings. With the new settings the update frequency is currently limited by the camera frame-rate.

3 Software Architecture

This Chapter covers the software architecture of the robotic arm. The architecture mentioned in this chapter forms the set-point controller of the robotic arm. The joint-level set-points are generated and send to the hard real-time controllers, mounted on the arm. The current software structure is analysed. From its results a new architecture is proposed. This change is proposed in order to facilitate the task stated in scenario one. This leads to an overall change in design and implementation.

3.1 Software architecture analysis

The software architecture was analysed to discover its structure and implementation. From the analysis in Appendix E several limitations were discovered. The software architecture uses the MoveIt motion-planning-framework (Sucan and Chitta, 2017). To connect to this framework a special Move_group interface is provided. One connection is sufficient to interact with the entire framework. However, multiple connections towards the same Move_group were initialized and used, making the structure unnecessary complex and computationally heavy.

The connections to the Move_group were realized in the *Task Planner*. The Task Planner is designed to sequentially plan and execute motions, which together perform a predefined task. Its architecture was not flexible towards changes or adding functionality. This was problematic, as no recovery procedures were implemented in case an error occurred, which is highly undesirable when designing autonomous systems. Another limitation was the inability to set velocity or acceleration parameters. Every motion was planned and executed using its maximum velocity and acceleration setting.

To summarize, the following limitation were analysed:

- The software is unable to recover from intermediate failures
- Multiple connections to the same Move_group interface are present
- The architecture is not flexible for changes or adding functionality
- Acceleration and velocity limits parameters cannot be set.

To resolve these limitations the entire *Task Planner* had to be restructured. Reconstructing the task planner leads to an overall change in architecture which is proposed in the next section.

3.2 Proposed Architecture

A new software structure is proposed to resolve the limitations stated in the previous section. A general overview is presented, each elements of this overview is described up to the Move_group. Everything after the Move_group works properly and does not require further explanation.

3.2.1 Overview

Figure 3.1 represents the proposed architecture. The different elements of this architecture are described below.

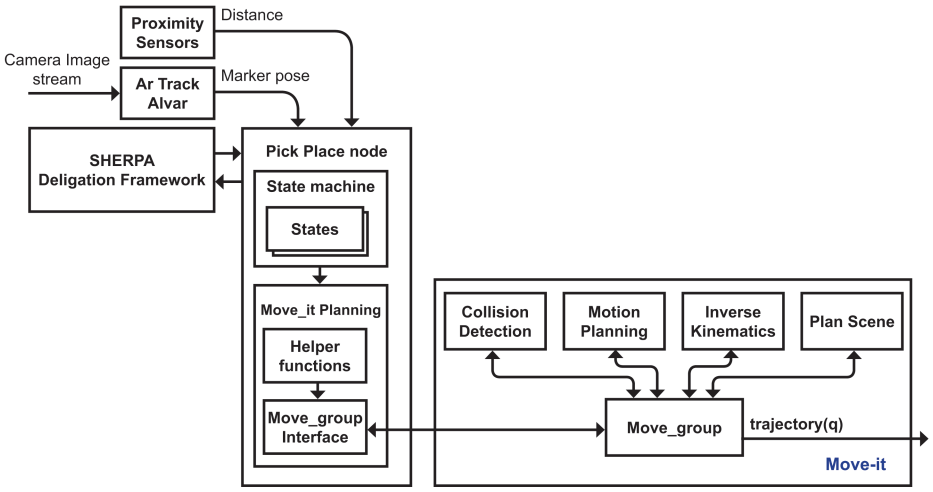


Figure 3.1: Proposed architecture

Sensors
One of the changes is the addition of an extra sensor. Besides vision data the measurements of a proximity sensor will included. This sensor is installed in the tip of the gripper and will be used during the last stage of grasping. When getting into close range of the UAV, the proximity data can be used to precisely lower the gripper into its interface, preventing damage to the UAV or robotic arm.

SHERPA Delegation framework
The SHERPA Delegation framework can be seen as a supervising layer in the overall SHERPA architecture. This framework is able to delegate tasks to the different actors in the SHERPA team.

Pick place node
The main functionality of the implemented functionality is towards grasping and docking an UAV, which comes down to a pick and place action and therefore a suitable name.

State machine
The sequential structure of the Task planner is replaced by a state machine. It is envisioned, that the use of a state machine enhances the nodes flexibility in the form of generalized states and tasks. It is expected that it will eases implementation or extension.

Move it planning
The Move_It_planning block is added to provide the interface for the Move_group and generalizes frequently used procedures. For example, the generation of a trajectory from its current configuration towards a given set-point and executing this motion. These functions also allow the adjustment of the speed and acceleration limits in MoveIt, which are taken into account when generating a trajectory or executing a motion. The Move_it_planning object can be reached from every state or task in the state machine.

Move it
The MoveIt framework is kept the same and does not need any modification. It has been added in this diagram to emphasis the single Move_group interface connection.

3.3 Pick Place node

In this section the underlying structure of the Pick place node is explained. A general overview of its structure will be provided to indicate how different elements are connected.

3.3.1 General architecture

A general overview of the Pickplace node is presented in Figure 3.2, in the form of a functional diagram. The different, of which it is composed, are described below.

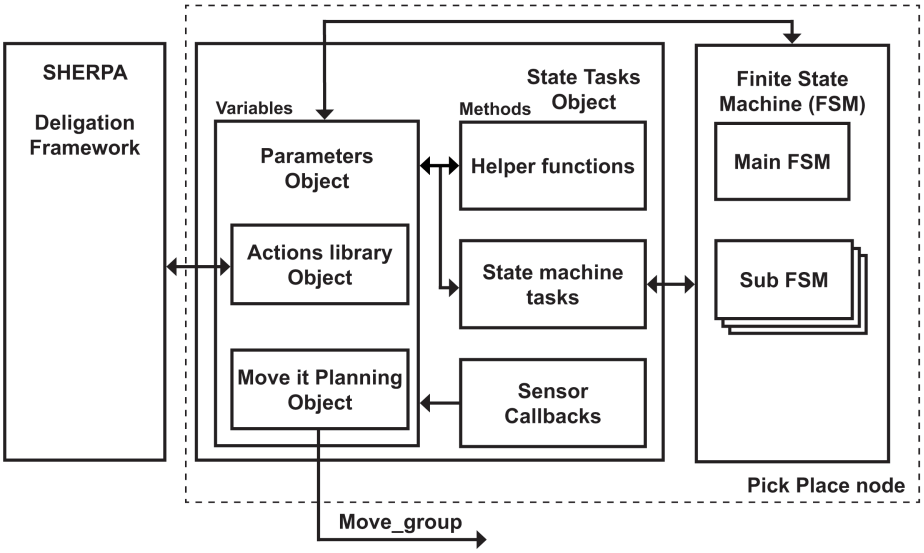


Figure 3.2: General structure of the Pick place Node

Actions Library

The connection to the SHERPA delegation framework is realized using the ROS action library. This library provides a standardized interface for interfacing with 'long-running' tasks that can be pre-empted if necessary (Eitan Marder-Eppstein, 2007). The action library knows three types of messages which can be user defined. In this architecture the messages are defined as follows:

Goal:
Goal or task request to execute. This message-type consist of the following parameters:
eventName: The name of the task to execute, for instance grasp. This name corresponds to a certain state within the state machine.
goalWaspID: Database ID name of a SHERPA UAV, for example Wasp0. This name is used to identify which UAV has to be retrieved.

Result:
progress: The percentage of the executed task.

Feedback:
progress: Progress in percentage of execution of a task
state: the name of the state that is currently active
task: Name of the task that was requested

Parameters

The parameters object, as the name suggests, contains the parameters needed throughout the state machine. It is able to communicate with *Helper functions*, *state machine tasks* and receives sensory data through callbacks. The advantage of this structure is that it allows to set certain parameters or flags, using states or sensory input, a "inRange" flag for instance. This flag is set when the proximity sensor is within detecting range and measuring and could be used to trigger certain states or transitions. Another advantage is the accessibility of the *Move_group interface* and the *Actions library*from every state or state task. Allowing control of the robotic arm and sending feedback information to the Delegation framework.

State Tasks

State-tasks bundles the *helper functions*, *state machine tasks*, *Parameters* and *callback functions* in one document, which centralizes the main functionality.

Finite State Machine

The Finite State Machine (FSM) contains the state machine structures. The *Main FSM* and several sub state machines are defined here. These *sub FSMs* are separately defined and could be started through the main FSM, creating a Hierarchical State machine or 'HSM'. The state machines are able to execute certain tasks defined in *State Tasks*. These tasks require a predefined structure in order to work with the state machine.

3.3.2 State machine package

For the implementation of the state machine architecture two ROS packages were considered. The *smach* package and the *decision making* package (Bohren, 2017; Cogniteam, 2014). Both packages offer similar functionality and provide a live graphical representation of the state machine and its active state, depicted in Figure 3.3. It was chosen to use the *decision making* package, as it is based on the C++ programming language, in contrast to *smach* which uses Python. C++ is the main programming language used in the project, so the *decision making* package is an obvious choice.

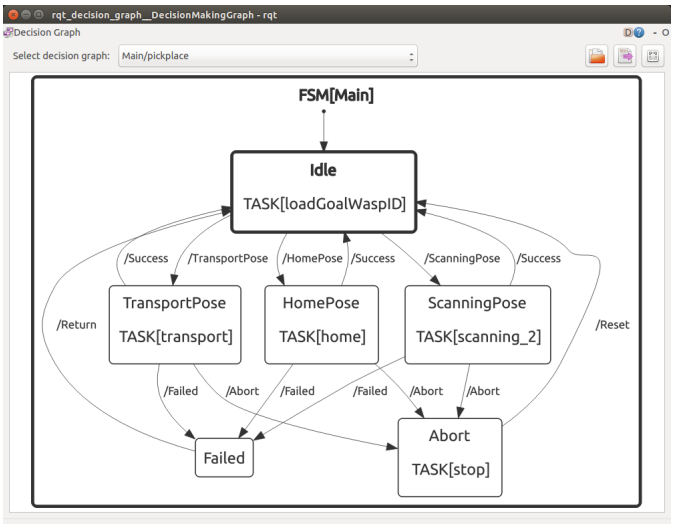


Figure 3.3: Visual representation of a state machine, generated with the decision making package

3.3.3 Main FSM architecture

The main state machine contains frequently used tasks or triggers sub state machine. This structure makes different states accessible for the delegation framework. Each state in the main state machine can be trigged using a task request. This creates a hierarchical structure that could be extended easily. A simplified representation of this architecture has already been presented in Figure 3.3. The entire structure of the main FSM is depicted in Appendix G.1. In this section the communication structure, starting from the delegation framework to the execution of a task is presented. An example of a task implementation is presented in the next section.

Task request
In rest, the state machine is in the *Idle* state. When an action is received, its corresponding state is triggered. For this system to work the *eventName* (set in the action goal) must be equal to its transition name defined in the state machine. In other words, the name in *eventName* must be equal to its transition name. In the diagram of Figure 3.3 these state transition name are indicated with a forward slash, for example */HomePose*.
When exiting the Idle state a task is triggered called *loadGoalWaspID*. This task checks if the received *goalWaspID* is known in a drone database . It searches trough a list of registered drones and compares their ID's. If a match is found the corresponding parameters are set. The list contains drone specific parameters like marker ID numbers, height, marker location and so on.

States
In the triggered state, a certain task is executed or sub state machine is started. When this task has been completed the system returns in Idle mode. When a procedure failed or is aborted the systems jumps to the corresponding state. In the abort state for example, any active motion is immediately stopped.

Tasks
Each task or intermediate step that has to be executed is defined as a task. These task can be implemented as general tasks, for instance *MoveToGoal*, which moves the end-effector of the SHERPA arm towards an end position defined.

3.3.4 Pick FSM

From the Main state machine other state machines can be triggered. These sub state-machines are separately defined and allows a hierarchical structure. The different stages of the first scenario are implemented as sub-state-machines and conveniently named *PickFSM*, *DockFSM* and *DeployFSM*. Explaining each of these sub state machines goes beyond the scope of this report. Instead the pick procedure is explained in more detail. The other state machines follow a similar procedure and are depicted in Appendix G.

The pick procedure actually covers two stages of scenario one, namely reaching and grasping. This is also represented in the flowchart of Figure 3.4. A full scale version of flowchart is presented in Appendix F. Each state could *fail* or *abort* the procedure, for clarity these states are depicted separately and not connected to each state. The two stages of scenario one are described below, as well as the last step in the procedure called *Lift*.

Reach
The first part of the pick procedure involves reaching the landed UAV. During this procedure the UAV is located within the work-space of the arm. The arm is put in its scanning position to scan the work-space, *ScanWS*. When the UAV is located, a collision object is loaded into the plan scene by the *Add Collision Object* state. The located UAVs position, with a certain offset, is set as Goal position for the end-effector, the end-effector is called Tool Centre Point (TCP) and is executed by the *Set TCP Goal* task. The offset is 10 cm in z-direction, above the gripper

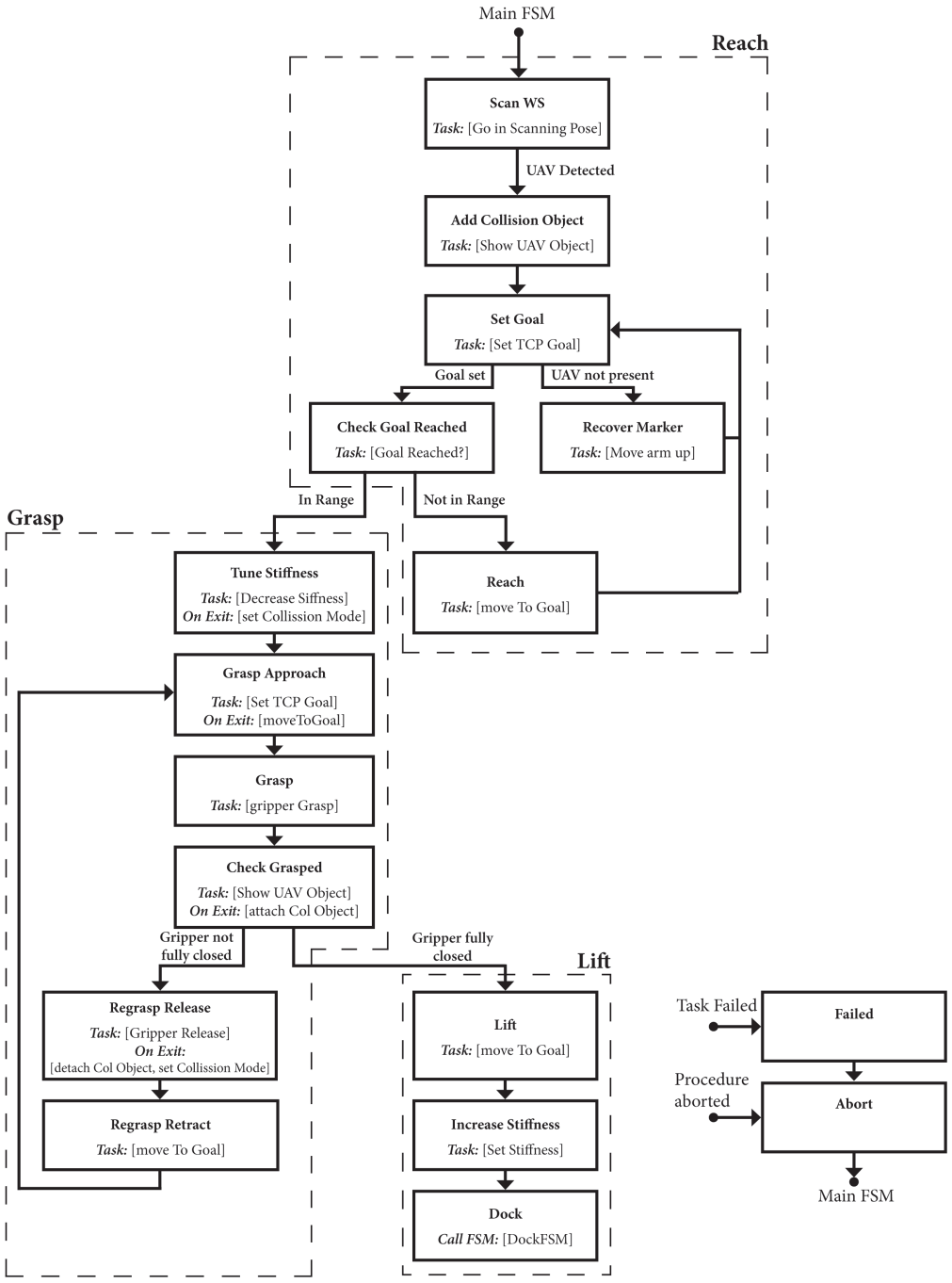


Figure 3.4: Sub state machine for picking up a UAV.

interface. As long as a marker (UAV) is detected the arm moves towards its goal position. The arms current and goal-position are compared in the *Check Goal Reached* state. If the UAV is not localized any more the arm moves upwards to re-localize it. Once the gripper is within range and still visible, the second part of the procedure is to *grasp* the UAV.

Grasp

The stiffness of the gripper joint is decreased to allow a smooth alignment of the gripper interface and the gripper itself. Secondly a collision between the gripper and the UAV is necessary during this operation. The settings are updated in the *Tune Stiffness* state In the *Graps Approach* state the TCP goal is set once more using the *Set TCP Goal* task. In close proximity of the

gripper interface (between 5 and 200mm) the proximity sensor data is automatically used as set-point for the z-direction. With the goal set the arm moves slowly towards its new position. At this point the acceleration and velocity scale is set to 10% of its maximum value. Once the new position is reached the gripper is grasps the UAV and checks the gripper conditions. These conditions are automatically updated by a helper functions called *GripperAngleCB*. When exiting the *Check Grasp* state the UAV is attached as a collision object to the arm in the plane scene, assuming the gripper is sufficiently locked. If the gripper is insufficiently locked, a retry procedure is executed. The gripper is released and the collision object detached in the plan scene. The arm is raised a couple of centimetres from where the *Grasp Approach* state is restarted. If the gripper is sufficiently locked the last part of the procedure is to *lift* and dock the UAV.

Lift
The last part of the procedure is to *lift* the UAV 5 cm of the ground. The stiffness of the gripper joint is increased to its maximum value and the docking procedure is started. This procedure is described in a separate state machine.

Failed & Abort
Each state could *fail* or *abort* the procedure.In the current implementation not all fail procedures are implemented. Any fail state that is not yet covered will abort the procedure and return to the *Main FSM*

3.3.5 Package limitations

During implementation several limitations were encountered or situations occurred that were not foreseen. For the latter the parallel execution of tasks was not foreseen.. When multiple tasks are triggered within a state, these tasks are executed in parallel. Meaning that sequential execution is not possible by simply listen multiple tasks. This forces the separation of states, one for each or these tasks. The positive side is that each task has to be executed correctly before proceeding to the next. If a task fails a reattempt for each individual step could be implemented. A disadvantage is that the size of the state machine increases rapidly.

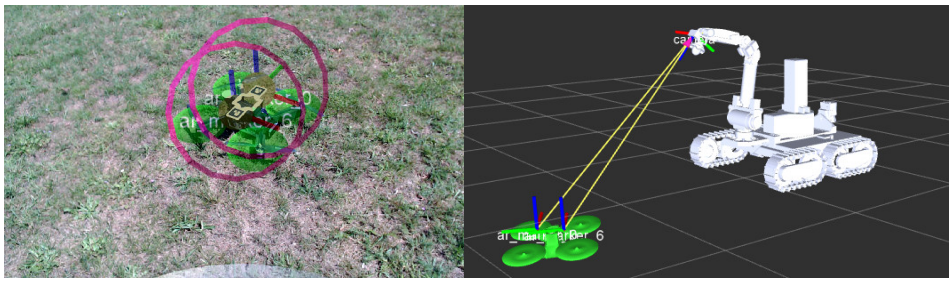
This problem became noticeable during the project, as more functionality and states were added, the larger the state machine became. The loading time of the graphical representation increased, up to the point the system was not able to visualize the state machine any more. The underlying Dot library was not able to generate the graphics due to its size or complexity. However, the state machine still functions without visualization.

4 Docking and deploying a landed UAV

In this Chapter the autonomous docking and deployment of a SHERPA UAV will be presented. The procedure is equal to the one described in scenario 1 and is divided in four stages. These stages are the localization, grasping, docking and deployment of the UAV.

4.1 Localization

When the task of battery exchange is delegated to the ground rover, the first priority is to localize the landed UAV. With the help of the maker-system, its relative position with respect to the rover is determined. The image feed of the camera is overlaid with the detected marker locations. This is indicated by the circles and reference frame in Figure 4.1a. As soon as the drone is identified, a virtual collision object is loaded in the plan scene and the camera image. The virtual drone, placed in the plan scene of the robot, is depicted in Figure 4.1b.



(a) Camera image with overlay **(b)** virtual plan scene

Figure 4.1: Virtual plan scene with still image (Barrett et al., 2018)

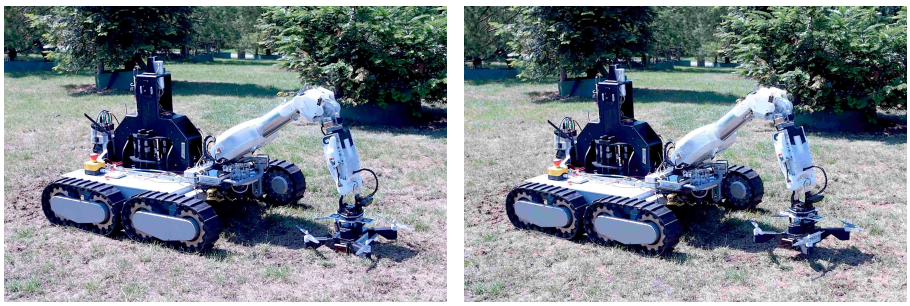
The pose of the UAV is send on to the rover. The rover plans a collision-free trajectory and approaches the UAV, such that the UAV is located within the workspace of the arm and can be grasped easily.

4.2 Grasping

When the rover has moved into position, the grasp procedure, as shown in Figure 4.2, is started. The gripper moves towards the UAV and stops when it is located within the grasping tolerances and located approximately 10 cm above the gripper interface. At this stage the plan scene is updated, such that collision between the gripper and UAV is allowed. Up to this point any form of collision is avoided. The compliance of the gripper's joint is increased for a smooth grasping procedure. Using the proximity sensor data the gripper is precisely lower into its interface. Once in position the UAV is grasped. By checking the end position of the grasping mechanism a secure lock is verified. In case the gripper is not locked properly, the lowering and locking procedure will be reattempted.

4.3 Docking

With the drone attached a collision free path is generated, towards a predefined position in front of the docking interface of the SHERPA-box. The gripper's compliance is increased to allow guidance during docking. The collision mode is updated, to allow collision between the SHERPA box and UAV. The UAV is slowly pushed in position. Once docked the SHERPA-box locks the drone and starts the battery exchange procedure. The docking procedure is depicted in Figure 4.3.



(a) Grasp the drone **(b)** Lift the drone

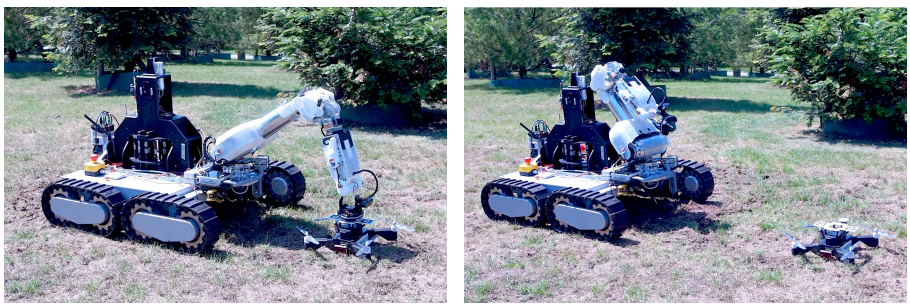
Figure 4.2: Grasping procedure

(a) Move to SHERPA box **(b)** Dock the drone

Figure 4.3: Docking procedure

4.4 Deployment

The deployment of a drone is in principle a reversed docking procedure. The drone is released and lifted from the SHERPA box. The drone is moved to a predefined position, located a couple of centimetres above the ground, behind the rover. The drone is gently lowered and released. Once the UAV is deployed, the arm is put into its transport position and the rover moves away from the UAV so that it can continue its mission. The deployment procedure is depicted in Figure 4.4.



(a) Deploying the drone back on the field **(b)** Move away from the drone

Figure 4.4: Deployment procedure

4.5 Conclusion

With the four stages completed it can be concluded that it is possible to autonomously pick-up, dock and deploy a landed UAV with the SHERPA arm, using vision based information. The implementation of the different hardware and software elements contributed to the achievement of this important task in the SHERPA mission. The results presented here are also used as part of a systems integration paper on autonomous battery exchange of UAVs (Barrett et al., 2018).

5 Tracking with the SHERPA arm

With the task described in scenario one completed, it is time to look at the possibilities of scenario two. In this scenario the goal is to grasp a drone while it is hovering. To do this the system should at least be able to follow the movements of a hovering drone, which will be referred to as 'tracking'. In this chapter a closer look is taken at the tracking possibilities using the robotic arm

5.1 Approach

Previous work on tracking a drone with a robotic arm was already executed. This research uses direct IMU data from the drone and applies data fusion with vision based position information, in order to determine the drone's position (Baarsma, 2015). However this approach is not applicable due to the current SHERPA framework.

Within SHERPA any information about other actors is managed by the Sherpa World Model or *SWM*. In this model, information is requested from the SWM. In this situation the arm would request IMU information of the drone to capture. The SWM requests this data from the drone and sends the received information back to the arm, as depicted in Figure 5.1. This system creates overhead and induces latencies. It is envisioned that the total latency is of such extent that the proposed method would not work, as it requires direct IMU information. The SWM is not available at this point to determine the actual latency caused by this framework.

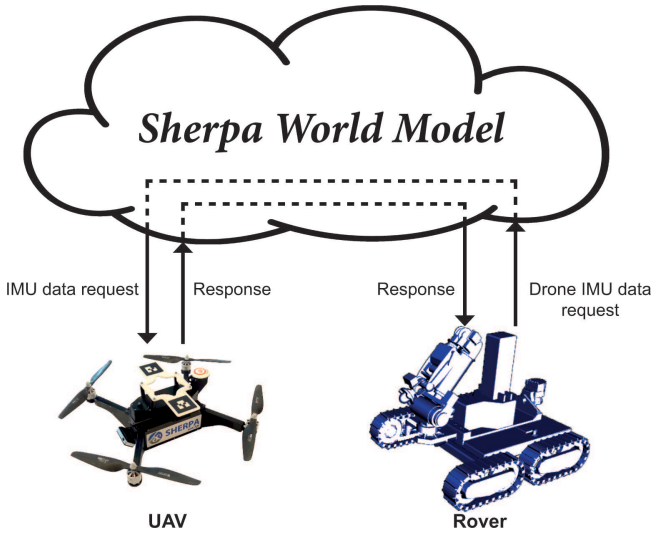


Figure 5.1: Sherpa World model framework

Requesting information through the world model is envisioned to induce high latencies, which is undesirable for tracking purposes. Instead both devices (drone and arm) are treated as two independent systems, meaning only the vision based information will be used for tracking. Some latency is still expected due to computations, however, expected to be rather small. To get some insight in these values the current setup is analysed.

For this research an AR Parrot 2.0 drone, equipped with a gripper interface, will be used for testing, as shown in Figure 5.2. The main reason to use an a Parrot drone is safety. The tests will be conducted in a controlled environment that can be monitored, in other words indoors. The SHERPA wasp is not equipped with propeller guards and propellers themselves are made

of carbon-fibre. This will create hazardous situations in case something goes wrong. The light weight AR Parrot drone with indoor hull is therefore much more suited.

Figure 5.2: AR Parrot 2.0 Drone equipped with markers and gripper interface

5.2 Analysis

The current hard & software is examined to determine the tracking capabilities of the robotic arm. The latencies in different parts of the platform are identified and the overall system settings are checked.

5.2.1 System latency

The task of following the movement of a hovering drone could be brought back to a few steps. Each of these steps require a certain amount of time and correspond to parts in the set-point control architecture. These parts are listed in table H.2 with their corresponding average latency.

System	Average latency
(1) Logitech C920 HD Pro	±100 [ms]
(2) AR Track Alvar	±30 [ms]
(3) Pick place node processing	±20 [ms]
(4) Arm Controller	±40 [ms]

Table 5.1: Different latencies in the system

The identification process of these latencies is covered in Appendix H. The total latency of the system is determined at approximately 190 milliseconds.

5.2.2 Hard real-time controllers

The robotic arm is designed to grasp and dock landed drones. The hard real-time controller are not tuned for fast motions because this not required when grasping a landed drone. The general velocity and acceleration limits of the systems are also kept relatively low for safety. In other words the SHERPA arm is never designed, tested of tuned for fast motions.

5.2.3 Sequential joint controller

The joint controller receives the trajectory set-points and sends them sequential to the hard real-time controllers. The communication speed towards these controller is approximately 40 Hz. In theory this is fast enough as the camera has an frame-rate of 30 Hz and should not be a major limitation. However the sequential nature of the joint-controller is. Each trajectory has to be completely finished before the next one is started. In principle an update of the current trajectories end position would be sufficient. To accomplish this a test was performed by emptying the position queue of the controller when a new trajectory is received. Unfortunately this resulted in unstable control and caused rapid vibrations. An explanation for this behaviour is that the previous trajectory was not finished, the newly received trajectory starts with a velocity of zero, meaning an abrupt stop of its current motion. With a set-point update frequency of 30 Hz this resulted in a shaking and vibrating robotic arm.

5.2.4 Conclusion

It can be concluded that the system is not suited for tracking an object in real-time. The arm is not configured or tested for fast motions and the combination of latencies and sequential position control are the cause for delays in the system.

Instead we could test its tracking capabilities by assuming low velocity displacements and by overcoming the delay in the system. For the latter trajectory prediction methods are proposed to compensate for the passed time.

5.3 Trajectory prediction

Trajectory prediction will be used to compensate for delays in the systems, with the goal of obtaining a more accurate real-time position. The accuracy of the prediction should at least be within the gripper interface margins, and preferably as accurate as possible. For the prediction methods two methods are considered, one being a linear extrapolation prediction method and the second a non-linear model based prediction method. Both approaches are described in more detail below. As a starting-point only the UAV's displacement in X,Y and Z is considered

5.3.1 Linear prediction

Linear prediction methods are characterized by its simplicity and capability of approximating a movement by piece-wise linear segments. Equation 5.1 described the linear prediction method, where $\mathbf{X}_{t0}$ denotes the position vector of the UAV at t_0 and $\mathbf{V}$ the velocity vector.

$$\tilde{\mathbf{X}}_t = \mathbf{X}_{t0} + \mathbf{V}(t - t_0) \qquad (t_0 \leq t \leq t_1) \tag{5.1}$$

The velocity is assumed to be fixed between t_0 to t_1 and is used to predict the next point with the help of its current position. In the rest of this report this model will be referred to as the *linear model*, or *linear extrapolation* model.

5.3.2 Non-linear prediction model

In a model based approach a dynamic model equation is often derived for the object at hand, in this case being an AR Parrot 2.0 Drone. The dynamic model can be used to predict its trajectory. However such a model requires real-time system information, like the force exerted by each of the propellers. The usage of real-time system information is something to avoid, to keep the systems separated and independent of each other.

Instead, the UAV will be approach as a mass-spring damper system. The UAV is seen as a mass, in an oscillating movement around a certain equilibrium position. This applies for each of the axis (X,Y and Z). The mass spring damper equation is presented in 5.2.

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{5.2}$$

With m denoted as the UAV its mass, damping factor c and spring constant k. By rewriting it into state-space form it becomes:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{5.3}$$

$$\mathbf{y}(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{5.4}$$

by solving this differential equation, with the UAV's current position and velocity as initial conditions, its behaviour can be calculated for a certain instance of time. By taking this time instance equal to the prediction time-step, the next point of its trajectory can be predicted. In the rest of this report this model will be referred to as the *prediction model*.

5.3.3 State variable filter

The prediction methods mentioned in the previous section require the current position x_{t0} and velocity v for prediction. From the vision system the UAV its position is known $\mathbf{x}_{t0}$ but its velocity is not. This velocity has to be be computed from the position information. Applying

the normal way of calculating , as presented in equation 5.5, the velocity will result in a noisy signal, as shown in Figure 5.4.

$$V = \frac{\Delta x}{\Delta t} \tag{5.5}$$

Another possibility is to reconstruct the velocity using a *state variable filter*.

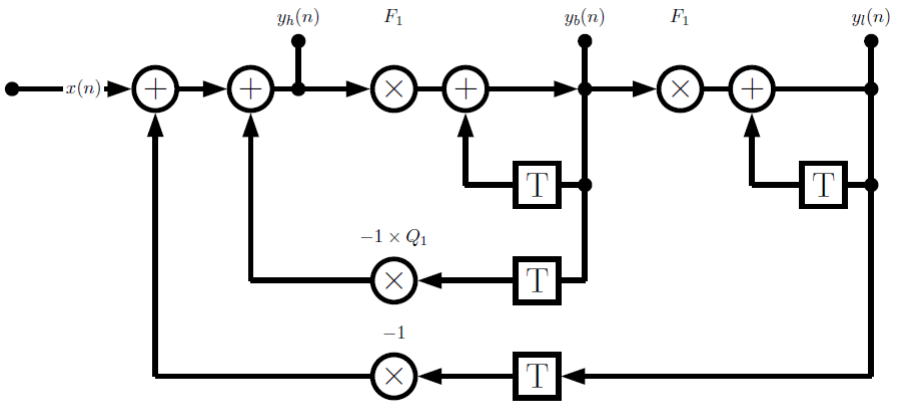


Figure 5.3: State variable filter (Cardiff.University, 2013)

The filter as shown in Figure 5.3 is characterized by a natural frequency ω_{nf} and a damping factor ζ_f. By choosing the filter's bandwidth to be larger than the drone its velocity, a good velocity reconstruction can be obtained (Conference, 1993). State variable filters are a popular use in tracking systems due to its underlying kinematic model. No specific parameters of the physical system are required (Swanson, 2011). For this reason, as well as low implementation complexity, the state variable filter is used.

Figure 5.4 depicts the difference of using the normal way of calculation and computation by the state variable filter. Only the displacement in x direction is depicted in this graph.

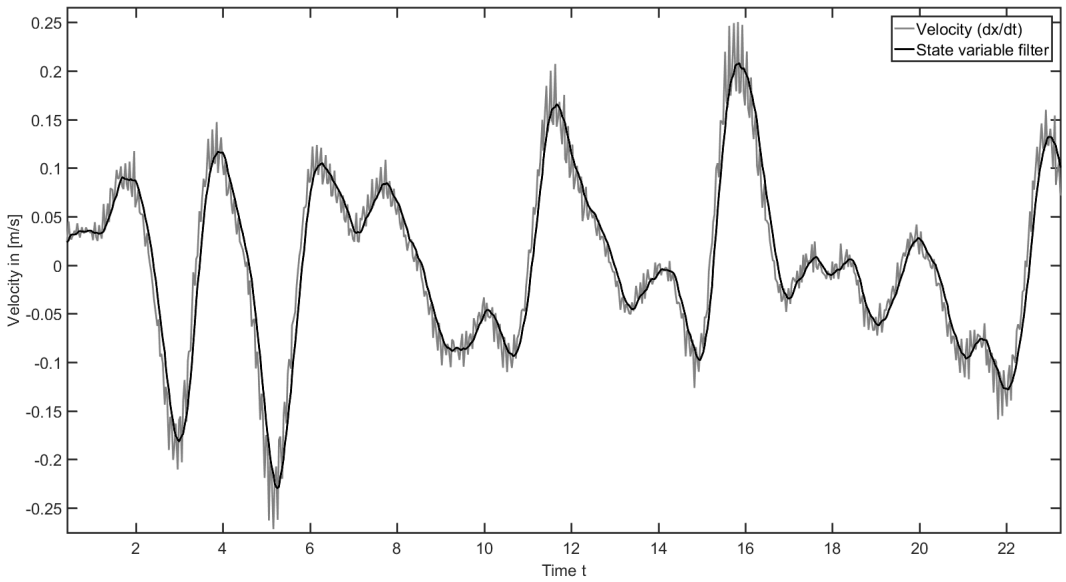


Figure 5.4: Velocity computed using the numerical differentiation versus state variable filter

The data-set for calculating the velocity has been obtained using the OptiTrack - Flex 3 Motion capture system (NaturalPoint, 2017), by recording the pose of a hovering drone in time.

5.3.4 Model performance

The performance of the trajectory prediction models are tested by comparing the error due to the delay in the system, with the error of the prediction models. These errors are calculated with respect the actual position of the UAV, measured by OptiTrack. The results are presented in the form of a histogram histogram, with a the normal distribution and standard deviation. The first plot in Figure 5.5 depicts the error caused by the delay in the system. The standard deviation (σ) is approximately 2 cm. In other words, the average error is around 2 cm. If we recall that the maximum tolerance of the gripper interface is 3 cm, one might argue that prediction methods are unnecessary. However, no external factors are considered during this experiment. In this context an accurate 'real-time' position estimate, of the UAV, is highly favourable.

The second plot in Figure 5.5 depicts the error of the linear extrapolation model. Using this model the standard deviation is reduced to approximately 9 mm, and therefore reduced a lot. The final plot depicts the error of the prediction model. Using this model, the standard deviation is even further reduced, to approximately 6 mm. The mean (μ) using this model is very low and can be stated to be zero, meaning the error distribution is well centralized around zero. From these results can be concluded that the prediction model shows the best results.

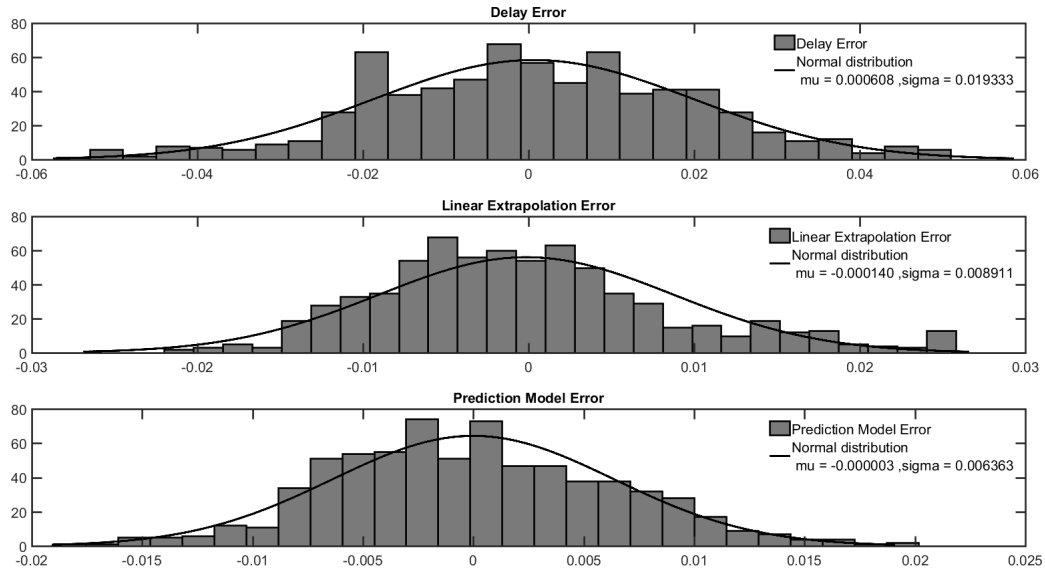


Figure 5.5: Error distribution using no prediction, linear extrapolation or prediction model

This comparison has been obtained by recording the movement of a hovering drone using OptiTrack. The received data is sampled down to 30 Hertz. The displacement in x direction is used to compute the results, assuming a delay of 200 milliseconds in the received signal. The delayed signal is realized by shifting the original signal 200 milliseconds in backward time. The delayed signal is used to compute the prediction results with a prediction step of 300 milliseconds. An extra 100 milliseconds is added to overcome the delay introduces by the state-variable filter.

The spring and damper parameters of the prediction model were manually obtained by comparing the mean and sigma values of different settings. The mass is determined by the total weight of the AR Parrot Drone. A parameter estimation approach has been considered for obtaining these values. Unfortunately no suited input parameter could be derived from the measurements in order to compute a parameter estimation.

5.4 Implementation

This section presents the overall implementation of the tracking functionality in the current platform.

5.4.1 General overview

The tracking functionality is implemented as a separate state machine. The general overview of this state machine is presented in Figure 5.6. Each state is described in more detail below.

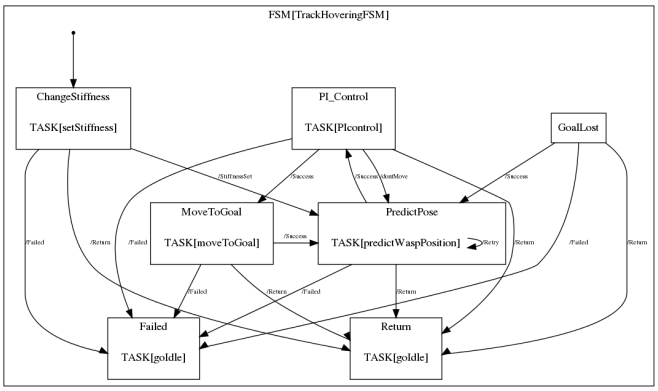


Figure 5.6: Track Hovering FSM, generated by the software

Change stiffness
When entering the state machine the first task is to increasing the compliance of the gripper joint. During tracking the camera will primary be faced downwards. By increasing the joints compliance less vibrations at the camera are expected due to movement of the arm. This allows the gripper to move more 'freely'.

Predict position
The predict position state actually performs multiple tasks, first the velocity is computed using a state variable filter. The computed velocity is used to predicted the uav its actual position. The parameters of the state variable filter and can tuned on the run, using Rqt dynamic reconfigure (Gassend, 2015).

PI Control
The next step is the PI Control state. This state has been implemented to be able to tune the predicted point. For instance by adding a PI controller, however this has not been implemented yet. Currently the gain is equal to one.

Move to goal
At this state the trajectory is computed and send to the Joint Controller using the *move To Goal* task. When the arm has reached its new position successfully, the system returns to the *Predict position* state and starts over.

Goal lost
In case the goal is lost, or in this case the drone, the system enters the Goal lost state. This state does not contain any functionality yet and is envisioned to scan the surrounding to find the drone.

Failed / Return
IF one of the states fails or a Return command is received the system switches to the corresponding state. In the current system both states return to the Idle mode of the main state machine. The fail state could be updated to start a recovery or retry procedure.

5.4.2 Prediction implementation

In the previous section it was mention that both prediction methods are implemented. The linear prediction method, as defined in equation 5.1, is easily implemented. For solving the differential equation a separate node is used. Solving the differential equation is achieved using the *ODE int* library. However this library caused errors during compilation when implemented directly in the current architecture, this occurred for unknown reasons. Testing this library in a separate node proofed to be successful. Therefore the *ODE int* library is implemented as a node called *ODE solver node*. This node solves the differential equation as defined in 5.2 upon a service request, and sends back the result. The mass spring and damper values and its initial conditions can be set when requesting the ODE solver service.

5.5 Prediction results

To determine the performance of the implemented prediction methods several test procedure are derived. In these test procedures the movement of a drone is recorded by two systems, the OptiTrack system and vision system on the ground rover. The latter is used to predicted the UAV position, to compensate for the delays in the system. In the performed test the OptiTrack measurement acts a 'ground trough' and is assumed to be the UAVs real time position. By applying the corresponding coordinate transforms, the OptiTrack measurement and prediction results can be presented with respect to the same reference frame. In these measurements the OptiTrack reference frame is chosen. The used test setup and the corresponding coordinate transforms are presented in Appendix I.

The following test procedures were derived:

1) **Manual/Fixed**
   The drone is moved in front of the camera by hand, with the robotic arm in a fixed position
2) **Hovering/Fixed**
   The drone is hovering in front of the camera with the robotic arm in a fixed position
3) **Manually/Tracking**
   The drone is moved in front of the camera by hand with the robotic arm tracking the drone
4) **Hovering/Tracking**
   The robotic arm tracks the movement of a hovering drone

1) Manually/Fixed

The first step in obtaining the test results is to compensate for possible offsets in the measurement setup. These offsets might be present due to misalignment of the OpitTrack markers. A static analysis is performed to identify these offsets. This is done by comparing the position of a static drone, measured by OptiTrack and the SHERPA vision system, both represented in the same reference frame. In this case an accurate static measurement of both systems is assumed.

Figure 5.7 depicts the position data of a drone moved by hand. The mean value of the first thirteen seconds of this measurement is used to identify possible offsets. In this setup an offset of $[-0.0241 - 0.0250 - 0.0108][m]$ in X, Y and Z direction are identified.

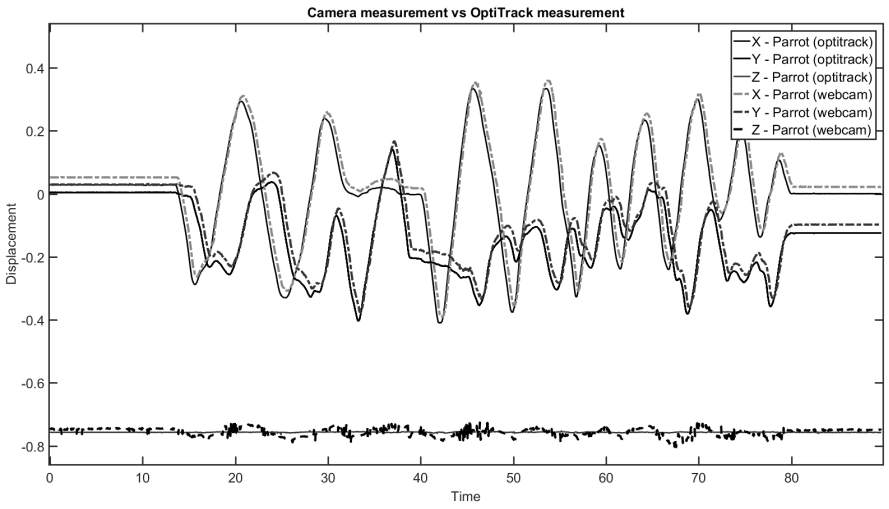


Figure 5.7: OpiTrack and vision measurement data of a drone moved by hand

The identified offset is subtracted from the OptiTrack measurement data, as shown in Figure 5.8.

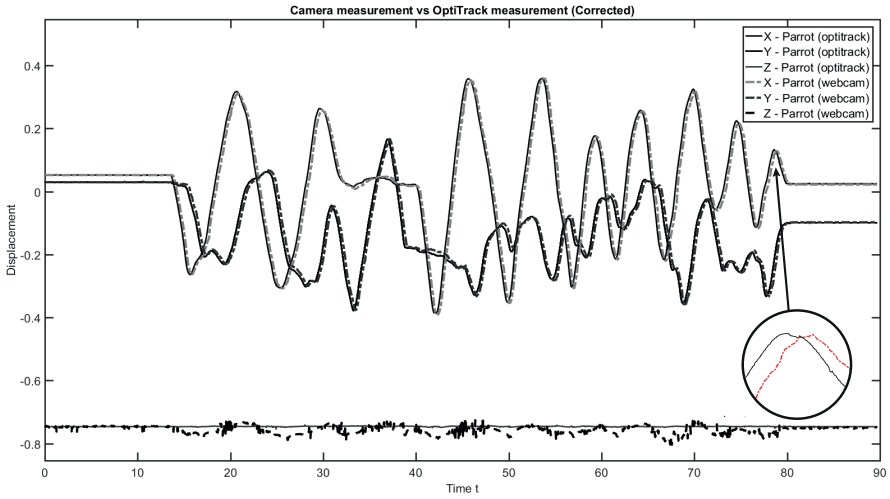


Figure 5.8: Adjusted OpiTrack and vision measurement data of a drone moved by hand

A very small displacement in time is noticeable in Figure 5.8. This displacement is appropriately 200 milliseconds, close to the identified 190 millisecond delay in section 5.2.1. However this delay is not consistent and varies from approximately 180 up to 300 milliseconds. Secondly no motions of the arm are executed during this experiment, so the arm controller delay should be neglected, resulting in an average estimated delay of 150 milliseconds. In ROS the usb_cam node is used to receive the camera images (Pitzer, 2016). This node gives a time stamp to each image captured by the camera. The time stamp is used during this experiment to monitor the overall processing time of the system. The time is measured up to the the point where the new set-point for the robotic arm is determined. Figure 5.9 depicts the measured processing time.

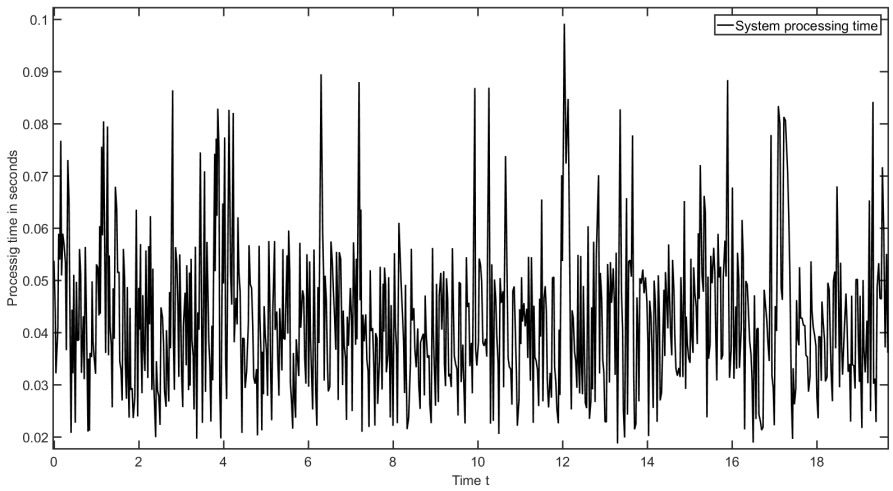


Figure 5.9: Processing time of the software

From the plot in Figure 5.9 it becomes clear that the processing time is indeed inconsistent, ranging from 20 up to 100 milliseconds. These values are still not near the 180-300 variation noticed in the graph of 5.8. Assuming a minimum delay of 160 milliseconds for the image capturing process is in contrast with earlier obtained results. The cause of this delay is unknown.

The prediction methods in this experiment use a fixed prediction step of 200 milliseconds. Therefore some variation between the prediction results and OptiTrack measurement is expected.

Applying the correct coordinate transform on the predicted data, results in the graph presented in Figure 5.10. This graph contains a close-up view of the results in x-direction.

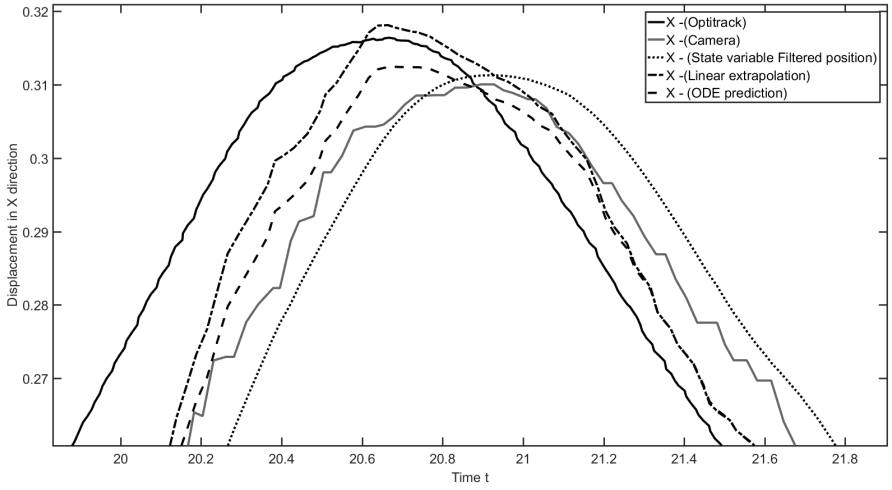


Figure 5.10: Predicition results

The camera data (in gray) is used as input for the state variable filter. Its output (dotted) is used for prediction. The prediction results are presented by the dashed and dashed/dotted lines. The linear extrapolation methods shows an overall better result with respect to the prediction model. This is surprising as the prediction model showed better performance. It is suspected that the lowered prediction step and variable processing latency are the cause for this mismatch. However, both prediction methods show an improvement with respect to the input value, but do not match the OptiTrack signal. An extra delay is introduced by the state variable filter, which is not accounted for during this measurement. The state variable filter requires three time steps to obtain a position and velocity estimate, these time-steps equal 100 milliseconds with a 30 Hz input signal. Meaning an even larger prediction step is required.

3) Manually/Tracking

During the experiments test procedure 3, Manually/Tracking, was also conducted. During this experiment an larger processing time was discovered. The processing time was ranging from up to 120 milliseconds with an average of 61.6 milliseconds. The extra delay is probably caused due to the computation of a collision free path by MoveIt. Secondly difficulties arose in measuring the drones position in OptiTrack, as shown in Figure 5.11. These are probably caused by the movement of the robotic arm above the drone. The same goes for tracking the movement of the camera by OptiTrack. Without an accurate camera pose no camera transform can be obtained, making comparison impossible.

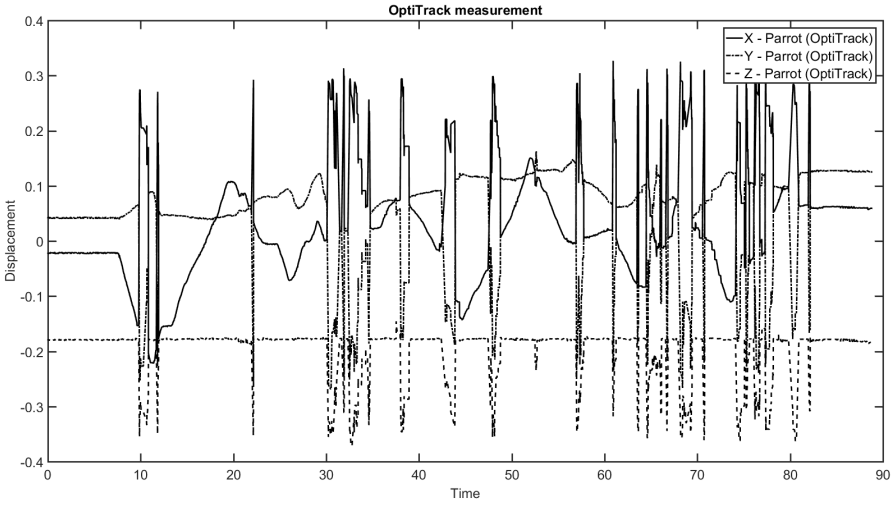


Figure 5.11: OptiTrack measurement of the drone position while tracking with the robotic arm

Unfortunately no further implementation or testing could be performed due to time constrains.

5.6 Conclusion

Two experiments for the tracking capabilities with the SHERPA arm were conducted. From the experiment with a manually operated drone and a fixed camera could be concluded that using prediction methods help in obtaining a more accurate position on the drones its 'real time' location. However the obtained latencies in Section 5.2.1 did not match the latencies observed during the experiments. Some of these latencies were accidentally not taken into account, other are caused due to irregular processing time or are simply unknown, and could not be traced directly.

It is expected that increasing the prediction time and including the varying processing time as a variable increase the accuracy of the prediction methods. This will probably also require retuning some of the current parameters used for the prediction model. Another improvement can be made by changing the procedure for obtaining the velocity estimate. The current implementation causes an extra delay of 100 milliseconds, increasing the overall delay by approximately 25%. A kalman filter is suggested because it only requires the current input measurements and the previously calculated state.

No accurate measurements could be obtained while tracking the movement of a manually operated drone. A more accurate measurement could probably be obtained by rearrangement of the OptiTrack cameras.

The SHERPA platform in its current state is not able to accurately follow the movement of hovering drone. Due to delays in the system the accomplishment of this task would be challenging, however further research is need.

6 Conclusions and Recommendations

6.1 Conclusions

The aim of this master thesis was to design and implement the autonomous grasping, docking and deployment procedure of a small-scale UAV, using a robotic arm mounted on the ground rover. For this task two scenarios were considered. In the first scenario, a safely landed drone has to be retrieved and docked on to the battery exchange station. The second scenario involves the tracking capabilities of the robotic arm, with the intend to grasp a hovering drone.

With the implemented marker-based vision system and updated task planner, the ground rover is able to autonomous grasp and dock a small-scale UAV on the battery exchange station. After the battery exchange procedure, the drone is deployed back in the field. In case the procedure or part of it fails, the system tries to recover from it by performing a re-attempt. For example, when a drone is not properly locked during grasping.

The second scenario could not be achieved with the current setup. The platform contains too high latencies for image capturing and overall processing. Secondly the entire platform is designed to operate on sequential position control and cannot be easily changed.

To increase the chance of successful tracking, the MoveIt software limits were adjusted to match the low level controller settings. In order to overcome the effects of the latencies a linear and model-based trajectory prediction method were considered. These methods were simulated using measurement data and afterwards implemented. However, could not be fully tested. During the experiments the actual drone position, obtained using a OptiTrack motion capture system, has been compared with the trajectory prediction data. From the test results can be concluded that the usage of these prediction models can contribute to obtain a more accurate 'real time' position.

6.2 Recommendations

For continuation of this project some recommendations are given.

Scenario 1:

- Extend the amount of recovery procedures to increase robustness.
- More feedback information towards the SHERPA delegation framework should be send.
- Implement a re-position procedure in case the UAV is not grasped correctly for battery exchange.

Scenario 2:

- Implement a high speed global shutter camera for better vision performance during tracking.
- Analyse the velocity and acceleration limits of the arm to allow faster motions.
- Switch to a variable prediction time step in order to overcome varying processing delays
- Change the velocity filter to reduce delay.

A Appendix: Background

In this appendix, general background information on the SHERPA platform is provided. Important hardware and software components with respect to this thesis are described

A.1 Hardware platform

A.1.1 The SHERPA arm

The SHERPA arm is a light weight 7 DOF (Degrees Of Freedom) actuated robotic arm, as shown in figure A.1 (Barrett et al., 2017). What makes this arm unique in comparison to industrial robotic arms is the usage of Variable Stiffness Actuators (VSAs)(Tan et al., 2017). These actuators allow for adjusting the mechanical stiffness of certain joint. These joints enables the adaptation of its dynamic behaviour for different tasks. One of these VSAs is implemented in the shoulder joint and another in the wrist. The seven degrees of freedom allows the robotic arm to fully fold into a transport configuration.

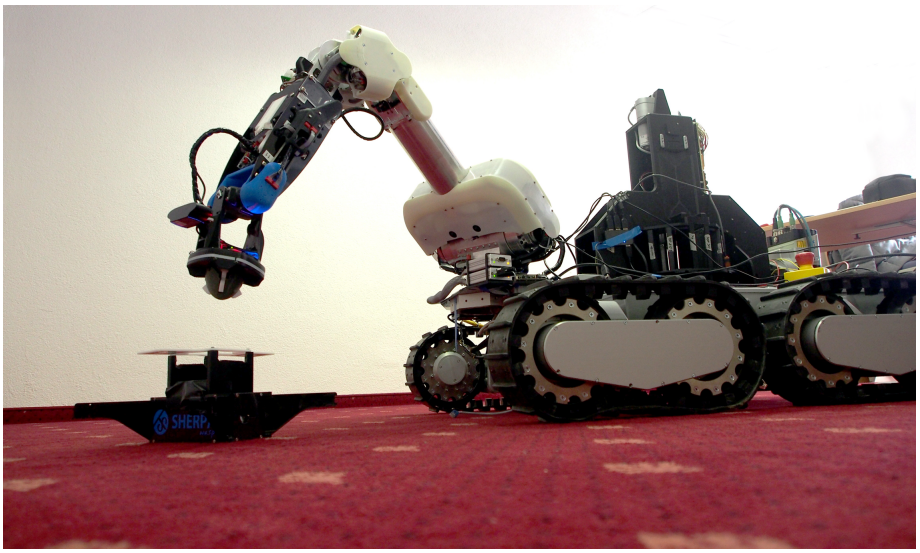


Figure A.1: SHERPA ground rover with 7 DOF robotic arm

The SHERPA arm is equipped with a custom designed gripper (Werink, 2016; Barrett et al., 2016). The gripper mechanism latches in an interface installed on the small-scale UAVs. The self-aligning design of the interface facilitates the grasping, which is easily drivable into the interface, ensuring a smooth grasping procedure.

A.1.2 Hard real-time control

The SHERPA arm is equipped with a total of eleven ELMO Whistle miniature digital servo drives that locally control the actuators (ELMO, 2017). The ELMOs control the actuators in position mode and receive feedback from the incremental motor encoders. Mechanical limit switches are directly connected to the ELMO drives to ensure safe operation within each of the joint limits. For feedback purposes each degree of freedom is equipped with a 14-bit absolute magnetic encoders which is connected through a Serial Peripheral Interface (SPI) with the sequential control. The communication towards the motor controller is realized via a Controller Area Network (CAN) bus system.

A.1.3 SHERPA-Box

The SHERPA box is a module designed for executing the battery exchange of SHERPA UAVs, depicted in Figure A.2. The UAV (1) can be docked on each side of the SHERPA-Box and is kept in place by a locking mechanism (2). The linear actuator (3) pulls the battery case downwards into the revolver mechanism (4). The revolver mechanism rotates to load a fully charged battery, which is pushed in by the linear actuator.

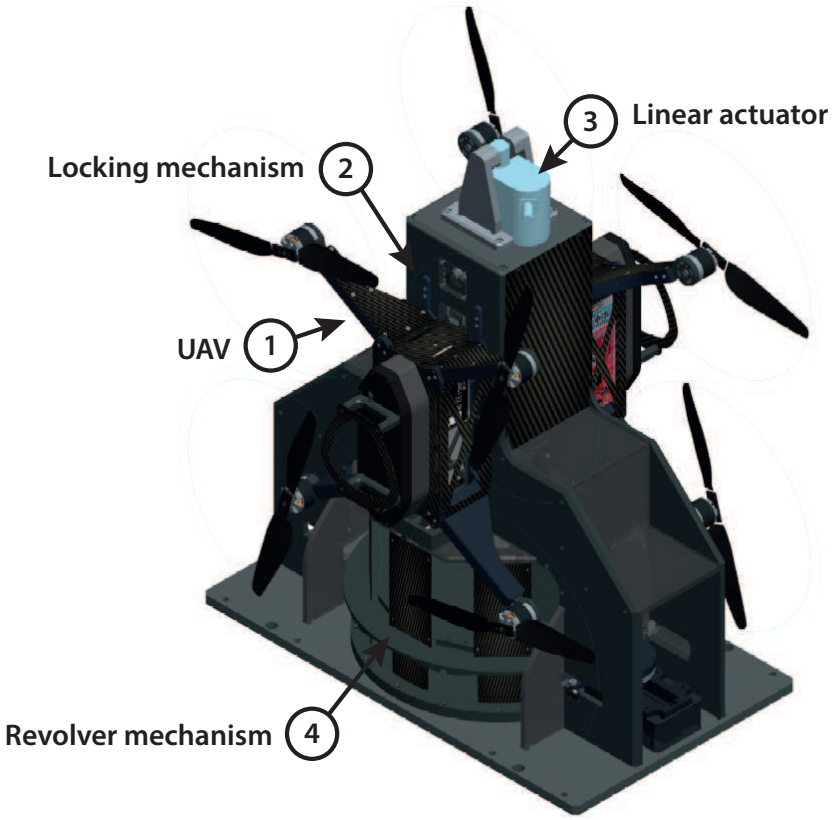


Figure A.2: SHERPA-Box, (AslaTech, 2017)

A.2 Software platform

The set-point controller is implemented on an Intel NUC i7 with Ubuntu 14.04 using ROS Indigo. ROS is an open source Robot Operating System and provides a platform for building robot applications (Quigley et al., 2009). Custom designed CAN and SPI interfaces are used to communication with the hard real-time controllers. Detailed information on the software architecture is provided in Appendix E.

B Appendix: Marker package comparison

B.1 Marker comparison

Table B.1 lists several criteria for comparing the different marker detection packages. More information the criteria is given below.

	Marker package			
Criteria	Ar Track Alvar (VVT Alvar)	Ar Pose (AR Toolkit)	April Tag (AptrilTag)	AR sys (ArUco)
Threshold settings	✓	✓	✗	✓
Binary decoding process	✓	✗	✓	✓
Automatic multi-marker detection	✓	✗	✗	✗
Bundle recognition	✓	✗	-	✗
Publishes Marker pose topic	✓	✓	✓	✓
Publishes tf frame	✓	✓	✓	✓

✓ Possible
✗ Not possible
- Unknown

Table B.1: Ros marker detection comparison

Threshold settings
The systems allows for changing threshold or tracking error settings through a the launch file or parameter settings.

Binary decoding process
The implemented systems uses a binary decoding technique for recognition. Ar Pose package for instance is able to read binary markers, however the implemented ARToolkit package uses template matching implementation techniques. This requires each marker design to be put in the system manually and configured.

Automatic multi-marker detection
The system is able to detect multiple ID numbers in a scene, without using predefined ID numbers or comparison lists for detection.

Bundle recognition
Multiple markers are combined and seen as one marker or object The markers centralize to a specific predefined point seen from each marker. The central point of the object will still me determined as long as at least on marker specified in the bundle is recognized. This systems is rather useful in situations when some markers might be occluded.

Publishes Marker pose topic
The estimated pose of an marker is published on a ROS topic

Publishes tf frame
A marker coordinate frame is published. This coordinate frame is often with respect to the camera frame.

B.2 Conclusion

Table B.1 clearly indicates that the *Ar Track Alvar* contains all of the comparison criteria. Especially the bundle recognition feature is an interesting one. Bundling multiple markers which are detected as one, increases robustness. In case a marker is occluded another might still be

within the field of view of the camera. Such a feature is especially useful when getting in relative close range of the UAV.

C Appendix: Camera specifications

C.1 VI-Sensor

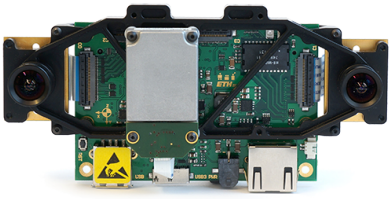


Figure C.1: VI-sensor

Resolution:	752*480 @ 30 fps
Stereo / mono:	Stereo
Image stream:	Monochrome
Interface:	Ethernet
Shutter type:	Global shutter
Dimensions:	Height x width x depth 57 mm x 133mm x 40mm
Weight:	130 g
Power supply:	12 15V <10W
Features:	IMU

Table C.1: VI-sensor specifications (Skybotix, 2014)

C.2 Creative Optia

Figure C.2: Creative Optia

Resolution:	640*480 @ 30 fps
Stereo / mono:	Mono
Image stream:	Colour
Interface:	USB 2.0
Shutter type:	Rolling shutter
Dimensions: (with clip)	Height x width x depth 57 mm x 80mm x 54mm
Weight: - With clip (fixed)	81 g
Power supply:	USB
Other features: -	

Table C.2: Creative Optia specifications (Cnet, 2007)

C.3 Logitech C920 HD Pro

Figure C.3: Logitech C920 HD Pro

Resolution:	1920*1080 @ 15 fps 1280*720 @ 30 fps
Stereo / mono:	Mono
Image stream:	Colour
Interface:	USB 2.0
Shutter type:	Rolling shutter
Dimensions: (without clip)	Height x width x depth 29 mm x 95mm x 24mm
Weight: - With clip - Without clip	 162 g 52.4 g
Power supply:	USB
Other features:	Automatic low-light correction, microphone

Table C.3: Logitech C920 HD Pro specifications (Logitech, 2017)

C.4 Kinect

Figure C.4: Xbox 360 - Kinect

Resolution:	640*480 @ 30 fps
Stereo / mono:	Stereo / depth
Image stream:	Colour & IR image
Interface:	Xbox360 kinect
Shutter type:	Rolling shutter
Dimensions: (without clip)	Height x width x depth 18.28mm x 304.8mm x 64mm
Weight:	1.36 kg
Power supply:	12V 1A
Other features:	-

Table C.4: Xbox 360 Kinect specifications (Andujar, 2017)

C.5 Bumplebee 2

Figure C.5: Bumblebee 2

Resolution:	648*488 @ 48 fps
Stereo / mono:	Stereo
Image stream:	Monochrome
Interface:	FireWire
Shutter type:	Global shutter
Dimensions: (without clip)	Height x width x depth 36mm x 157mm x 47.4mm
Weight:	342 g
Power supply:	12V 2.5W
Other features:	-

Table C.5: Bumplebee 2 specifications (Integrated Imaging Solutions Inc, 2017)

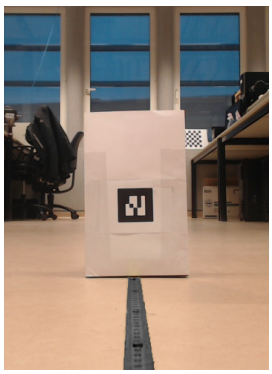
D Appendix: Vision test results

D.1 Measurement setup

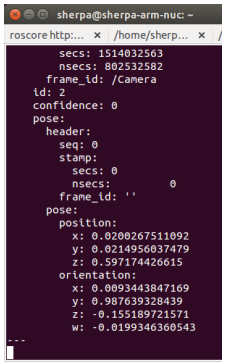
A test is preformed with the chosen vision system to determine its accuracy. The results of these test should be within the tolerances stated in requirement 1 of section 2.1. For these tests a marker with a size of 44x44 mm was used

To determine the accuracy of the vision system a test is preformed to compare the measured distance of a marker with respect to its actual position. A second test is preformed to determine the minimum angle of the marker which is necessary to detect the marker.

Distance & accuracy
To determine the accuracy of the vision system a marker is placed manually on a measurement tape, with an interval of 20 cm, Figure D.1a. For each position the output of AR Track Alvar is noted, depicted in Figure D.1b. This test is preformed twice. Once using the camera its full resolution of 1920x1080 pixels and once using a resolution of 1280x720 pixels. This measurement was preformed twice because the full resolution measurement was computationally heavier, this reduced the overall performance of the entire system.



(a) Marker as seen from the camera



(b) Measurement output of AR Track Alvar

Figure D.1

Marker angle & Detectability
The second test is to determine the minimum angle required to detect a marker from a certain distance. For this test the marker is put in a predefined angle, as shown in Figure D.2. The marker is placed on a measuring tape with an interval of 25 cm. For each interval the detectability of the marker was noted. In other words if the marker could be detected or not.

Figure D.2: Marker detection in combination with the robot model

D.2 Distance & accuracy measurement results

Actual Distance [cm]	Measured distance [1280 x 720]	Measured distance [1920 x 1080]
20	20.1	20.0
40	39.6	38.9
60	59.7	60.0
80	79.6	80.1
100	99.9	100.0
120	120.6	120.2
140	140.9	140.6
160	161.3	160.0
180	182.4	182.9
200	203.1	201.6
220	223.1	222
240	235~242	242
260	257~260	264
280	286~290	284.6
300	300~311	307
320	-	323
340	-	342
360	-	363
380	-	385
400	-	408
420	-	416
440	-	-

Table D.1: Use caption to elaborate

D.3 Marker angle & detectability measurement results

Distance: [cm]	25	50	75	100	125	150	175	200	225	250	275
Angle: [°]											
5°	✓	✓	~	✗	✗	✗	✗	✗	✗	✗	✗
10°	✓	✓	✓	~	✗	✗	✗	✗	✗	✗	✗
15°	✓	✓	✓	✓	~	✗	✗	✗	✗	✗	✗
20°	✗	✗	~	✓	✓	~	✗	✗	✗	✗	✗
25°	✓	✓	✓	~	~	✓	✓	✗	✗	✗	✗
30°	✓	✓	✓	✓	✓	✓	~	~	~	~	✗
35°	✓	✓	✓	✓	✓	✓	✓	~	~	~	✗
40°	✓	✓	✓	✓	✓	✓	✓	✓	~	~	✗
45°	✓	✓	✓	✓	✓	✓	✓	✓	✓	~	✗

Detected	✓
Unstable	~
Not detected	✗

Table D.2: Use caption to elaborate

E Appendix: Software Architecture Analysis

This Appendix described the analysis of the set-point control architecture of the robotic arm.

E.1 Current architecture

The current control architecture is presented in Figure E.1. This control architecture is composed of several blocks, namely a *Task Planner*, *Path Planner*, *Joint Controller*, *State Observer* and the *Plan Scene*. Some parts of this diagram were already envisioned, however never (fully) implemented and are indicated by the dashed rectangle.

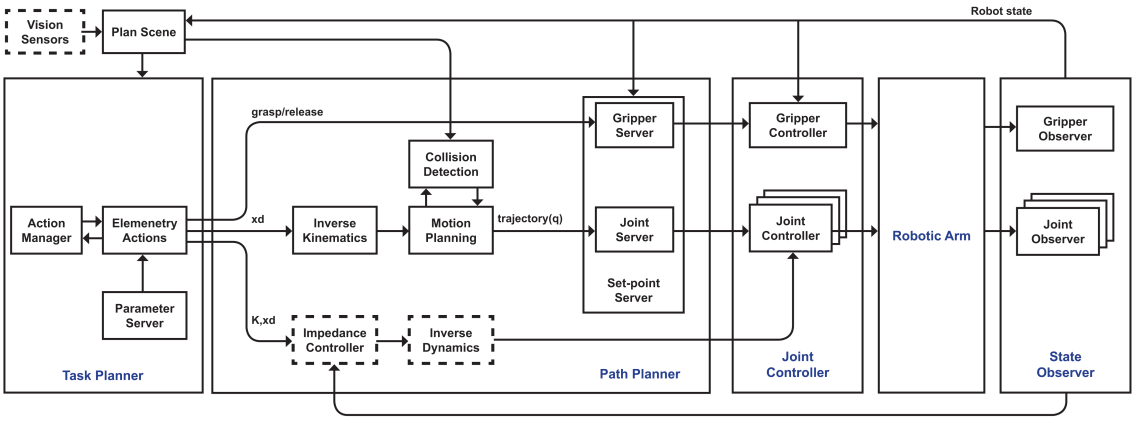


Figure E.1: General control architecture of the SHERPA arm, adapted from (Barbieri, 2016)

Task Planner
The Task Planner handles and monitors the sequential execution of motions which combined preform a certain task or *Elementery action*. An elementry action request, like grasp the UAV, is received through the *Action manger*, who also monitor the tasks progress. The *Elementery Actions* package generates the set-points of each task and sends them to the *Path Planner*. General system information is received trough the ROS parameter server.

Path Planner
The path planner generates a collision free path using a sampling-based algorithm and creates the set-points for the *joint controller*. An inverse kinematic model of the SHERPA arm is used to calculate the set-point for each joint. An array of set-points or trajectory is send to the joint-controller who executes them in a sequential order.

Joint Controller
The joint controller is implemented as a Proportional-Integral feedback controller and sends the joint set-points to the low-lever controllers. The feedback signal for this controller is received from the *state observer*

State Observer.
The state observer monitors each of the absolute encoders within the robotic arm and sends the current configuration or "state" of the robot arm back to the high level control.

Plan Scene
The plan scene can be seen as a virtual map containing information of the robots and its surroundings. This information is taken into account during *motion planning* to prevent external and self-collisions.

E.1.1 MoveIt & Move groups

The software structure as presented in Figure E.1 gives an global functional overview of the entire system, however does not represents it correct. Multiple parts of this architecture are actually implemented using *MoveIt*. MoveIt provides an easy-to-use platform for developing advanced robotics applications (Sucan and Chitta, 2017). The *Move_group Node* or simply Move_group connects the different elements and makes them accessible through the *Move_group interface*. Via this interface set-points, trajectories, settings and so forth can be send to the Move_group. The Move_group handles the path planning, collision detection and execution of any settings or requests. Visual feedback of this process can be provided using for instance RVIZ. An example of such visual feedback has already been presented via Figure 2.2 of the previous chapter. A more accurate overview of the actual software architecture is depicted in Figure E.2.

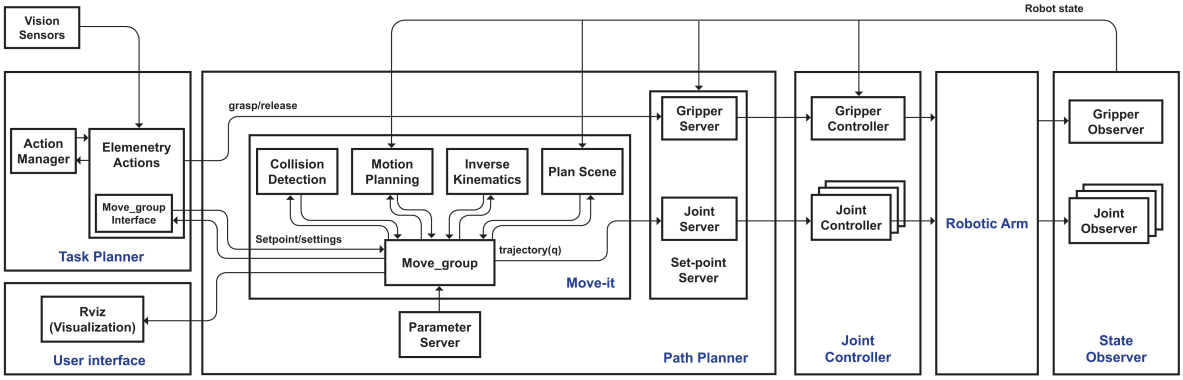


Figure E.2: Accurate overview of the software architecture

As a next step a closer look is taken at the Task planner and its connection with the Move_group.

E.1.2 Current Task planner & its limitations

Figure E.3 presents a global overview of the Task planner in combination with the Move_group. A description of the current Task planner elements is given below.

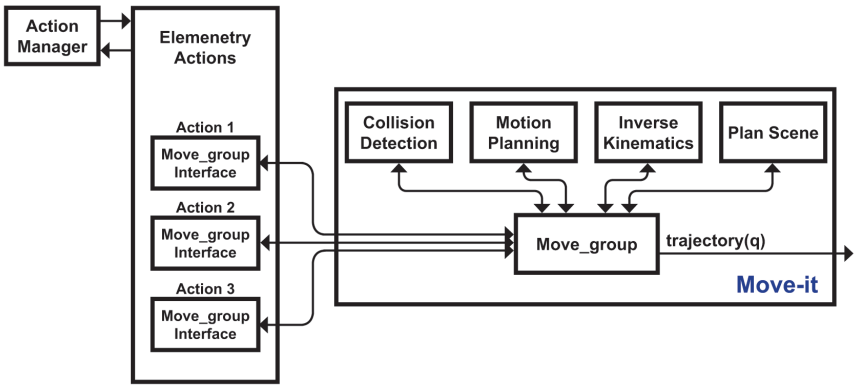


Figure E.3: Current Task planner

Action Manager

The action manager is a node that sends tasks or *Actions* to the Elementary actions packages and monitors its process if completed or not.

Elementary actions node
The Elementary actions node receives requests from the action manager to preform certain *Tasks*. These task are predefined within this node. Each of these consists of a number of sequential steps necessary to preform a task, like grasping. If any of the intermediate steps fails, the entire procedure fails without trying to recover the failed step. Without any recovery or reattempt procedures the system is rather sensitive to failures. A more flexible way of defining tasks with the ability to monitor and recovering from failed steps is highly desirable.

When looking at the general overview of this node it was noticed that each of the tasks interfaces with the same Move_group separately. From an architectural point of view this kind of implementation isn't very pretty and makes it computational speaking unnecessary heavy. A more elegant solution would be to interface through only one Move_group connection. Internally each task is connected to the same Move_group interface.

Another important feature is the ability to set velocity and/or acceleration constrains, which is not possible at the moment. This means that every motion is executed at full speed and acceleration within the software. To prevent any accidents the speed and acceleration limits are tuned down in the low level controllers. The control of the speed and acceleration is especially important when preforming delicate or precise operations, like docking the UAV.

To summarize, the following limitation were discovered within the current architecture:

- Unable to recover from intermediate failures
- Multiple interfaces to the same Move_group
- Architecture isn't flexible for changes or adding functionality
- Inability to set acceleration or velocity limits

To resolve these limitations the entire node has to be restructured. To do so a different architecture is proposed in the next section.

F Appendix: Pick State Machine

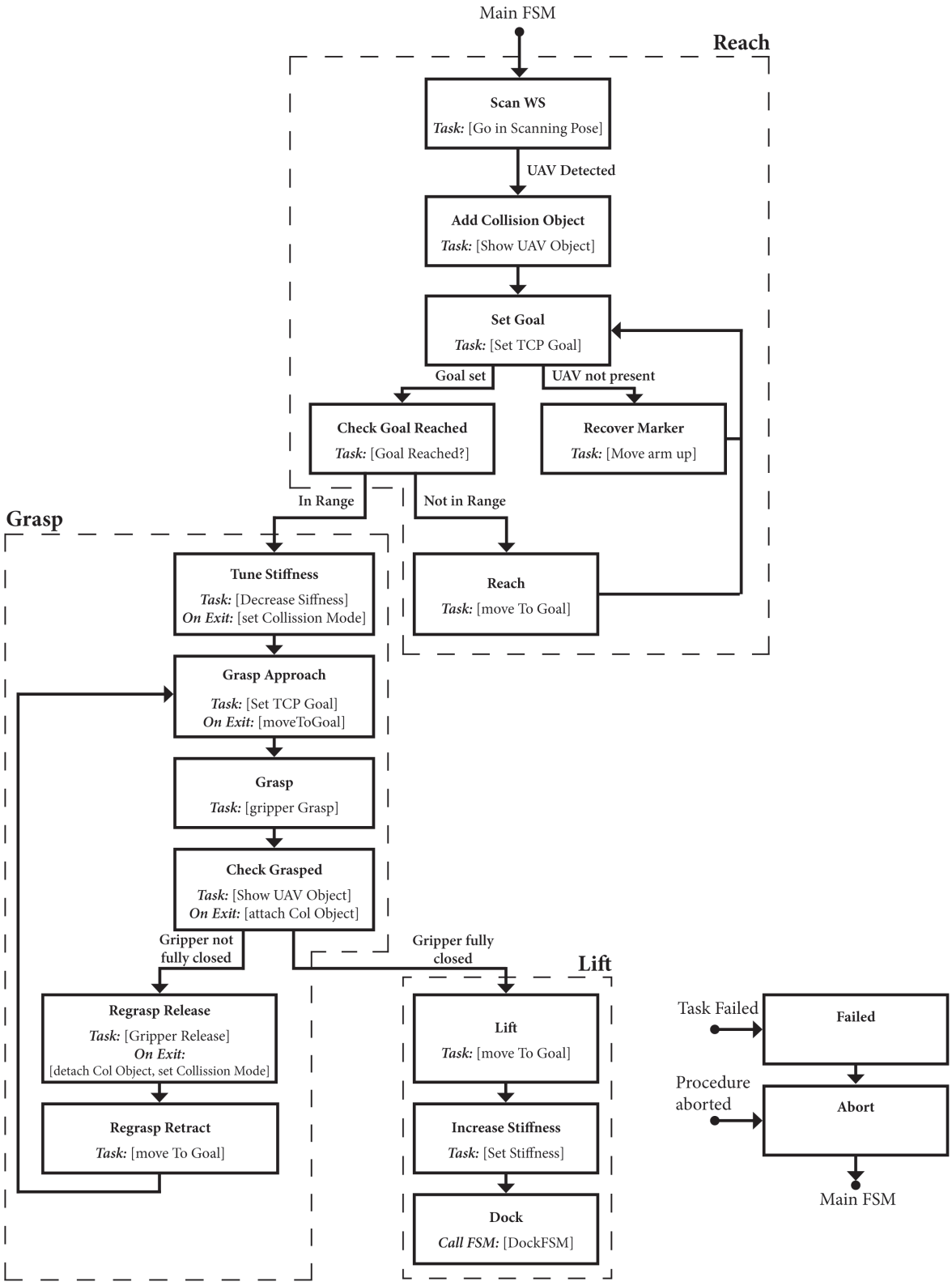


Figure F.1: Pick procedure

G Appendix: State Machines

G.1 Main FSM

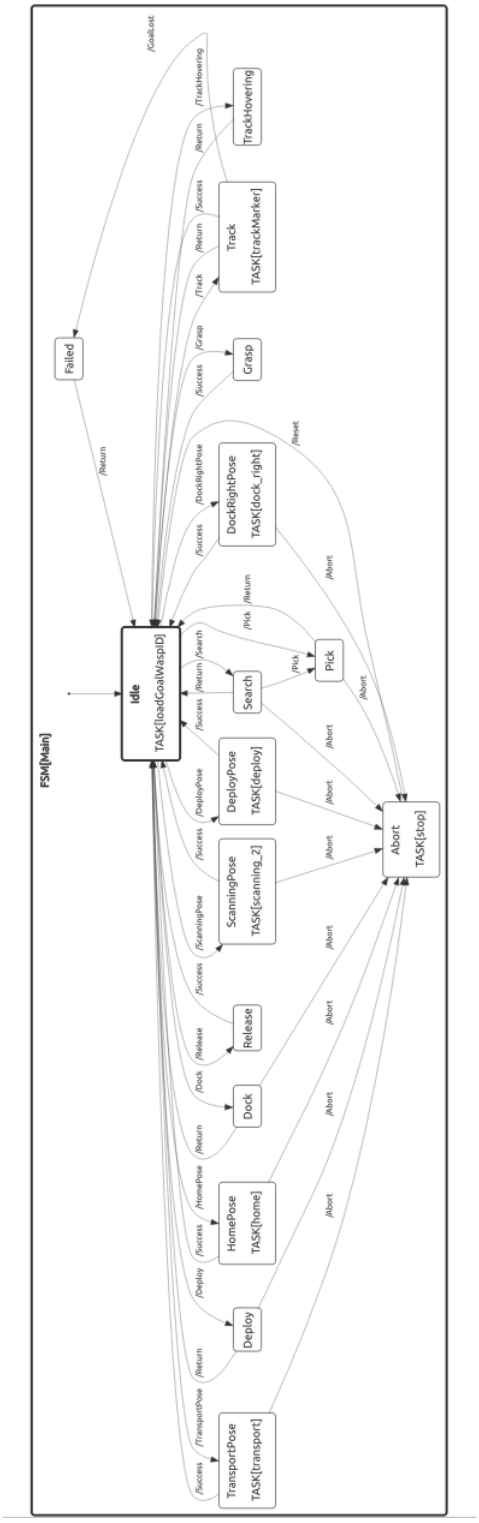


Figure G.1: Main FSM

G.2 Pick FSM

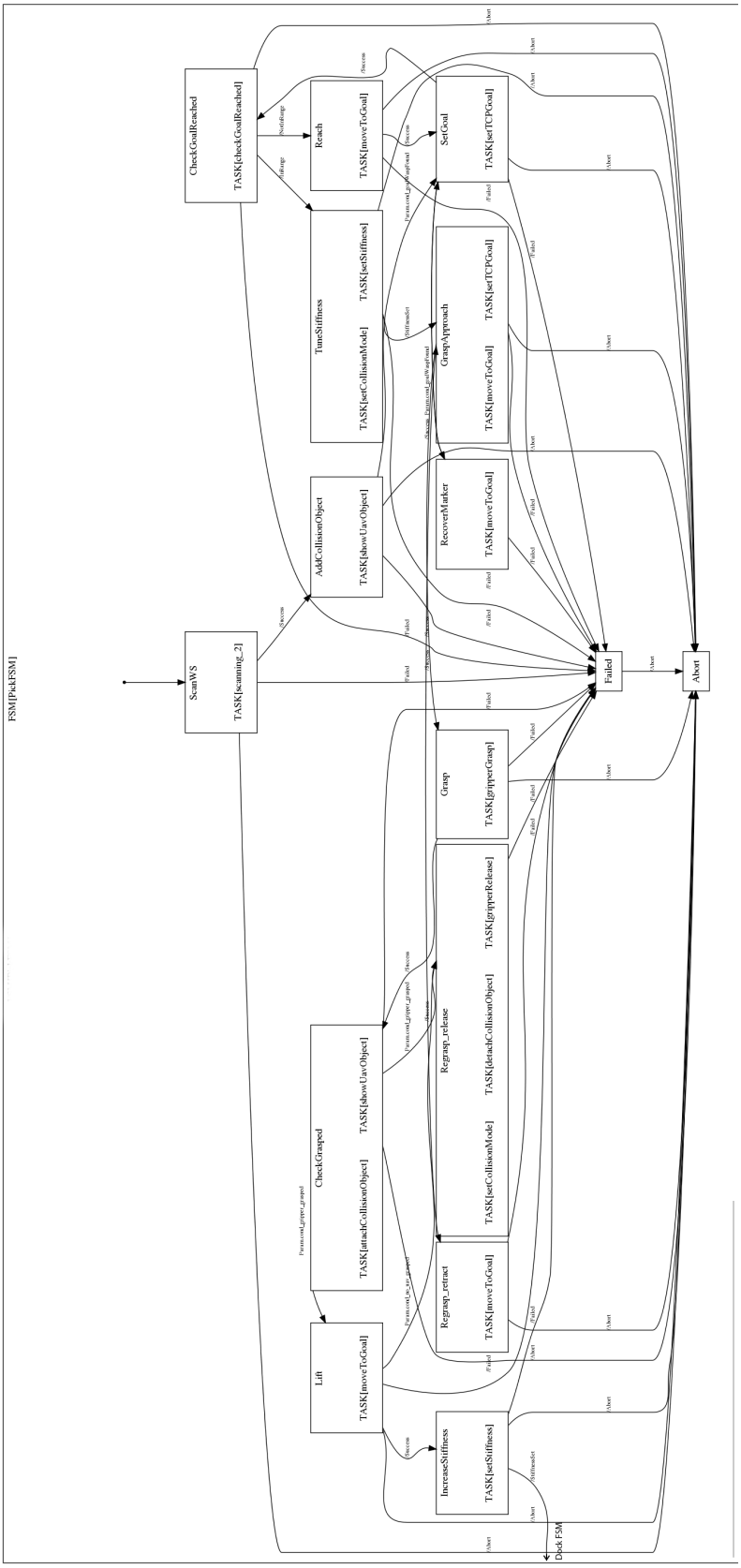


Figure G.2: Pick FSM

G.3 Dock FSM

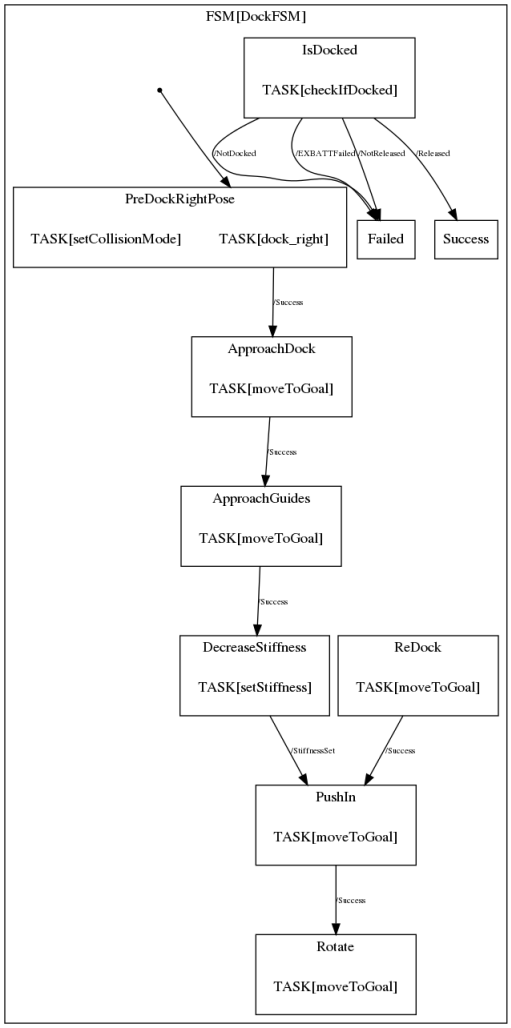


Figure G.3: Dock FSM

G.4 Deploy FSM

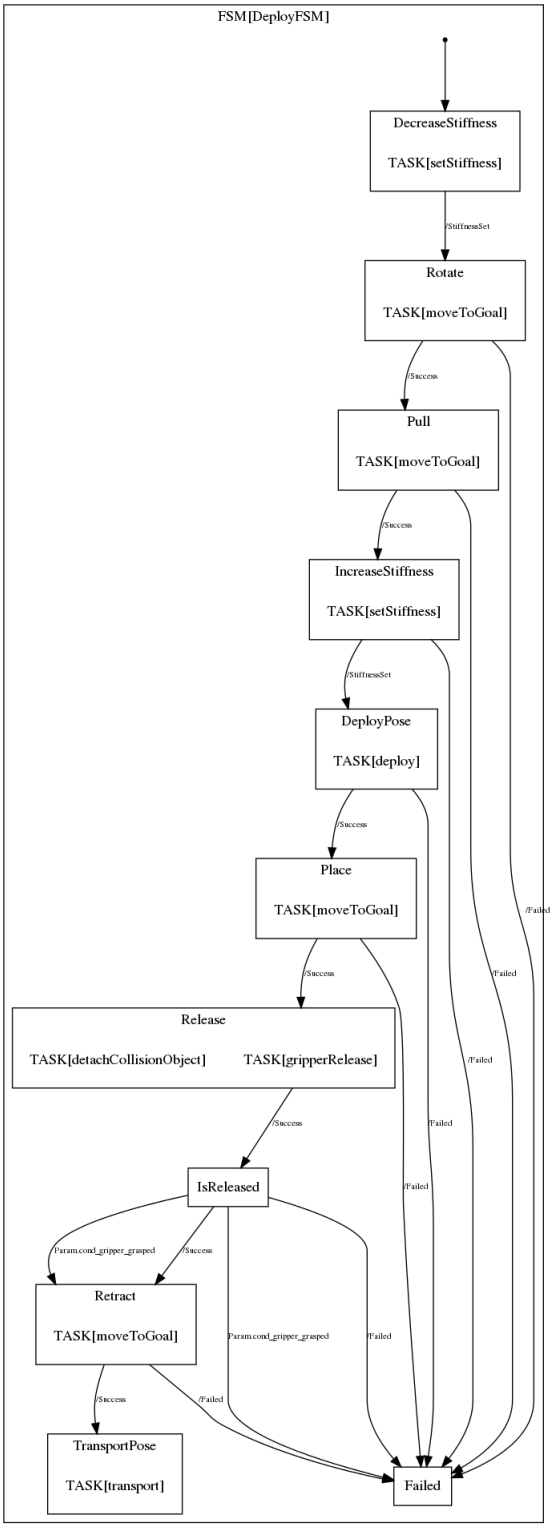


Figure G.4: Deploy FSM

G.5 Search FSM

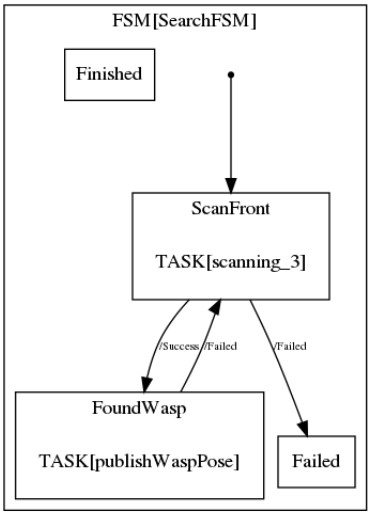


Figure G.5: Search FSM

H Appendix: System latency test

H.1 Latencies

The latencies in the system can be divided into three main topics, *image capturing*, *processing* and *controlling*. The first one concerns the time it takes to receive and captured an image, the second, the time to process the image and generate a set-point for the end effector. The latter consists of the time it takes to send the set-point to the arm and execute the motion.

H.1.1 Image capturing

In image capturing process the refresh rate of a camera is generally given, in this case 30 Hz. However the time it takes to process this data and send it to the computer is not. To get an idea of this latency a measurement setup is used as depicted in Figure H.1.

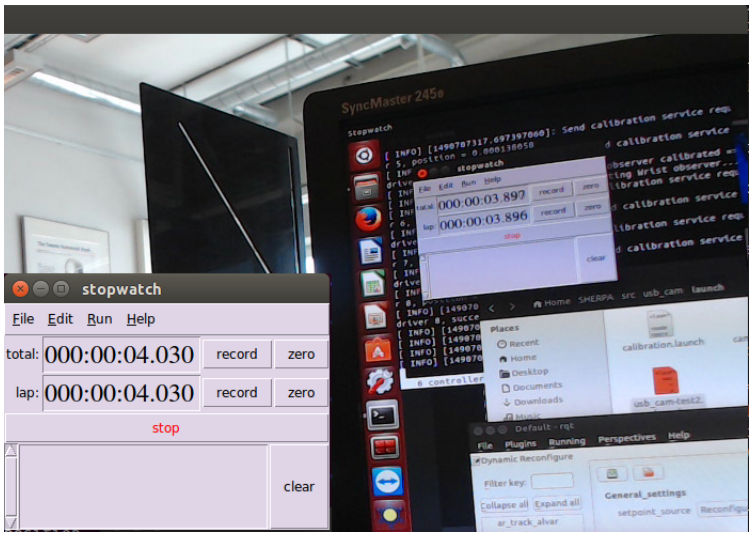


Figure H.1: Measurement set up to determine the latency in the image capturing process

In this set-up the camera is pointed towards a monitor and the received image is displayed on the screen. A timer is started on the screen and taking a screen-shot the actual time of the timer is shown as well as the time in the received camera image. The difference between the actual time and the time in the received camera image is caused by the latency in the image capturing process. The average latencies are shown in table H.1.

Camera	Average latency
Creative Optia	±100 [ms]
Logitech C920 HD Pro	±100 [ms]
VI-Sensor	±80 [ms]

Table H.1: Image capturing latencies using different devices

From the measurement results shown above it could be noted that both web-cams have a latency of ±100 [ms] and use an USB 2.0 connection. The VI-sensor is slightly faster with a latency of ±80 [ms] and uses and Ethernet connection. For compassion a slightly newer Logitech C922 with a USB 3.0 connection was tested. This measurement resulted in a latency of ±100 [ms]. This is a bit surprising as the systems hardware supports USB 3.0 devices, and therefore expected to have a lower latency. It is assumed that the latency is caused by the internal bus or

bus drives, but was not further examined. The Logitech C920 HD was chosen to be mounted on the arm, so 100[ms] will is used for the image capturing latency.

H.1.2 Processing -Pick place node

The latency in the Pick place node is measured by using the ROS time variable. When a new measurement is received from AR Track Alvar the current ROS time is stored. The received measurement it processed by the node until a new set-point for the arm is calculated. At this point the stored ROS time is subtracted from the current ROS time, resulting in the processing time. This time is printed in the terminal window. Using this method an average processing time of 20 milliseconds is obtained.

H.1.3 Processing -AR Track Alvar

In ROS the usb_cam node is used to receive the camera images (Pitzer, 2016). This node gives a timestamp to each image captured by the camera. This timestamp is passed along within AR Track Alvar to the pose estimate output (topic: ar pose marker). So each measurement published by AR Track Alvar contains the timestamp of the received camera image. This timetamp is subtracted from the current ROS time when a new measurement is published, hence resulting in the processing time required from a received image up a pose estimate output. An average latency of 30 milliseconds was determined

H.1.4 Arm controller

The latency due to control process can be measured by sending a set-point to a certain joint and record the time it takes to reach the set-point position. The measured time is naturally dependent on the step-size of the set-point. Once near the UAV, in line of sight suited for tracking, the changes in joint set-point are expected to be relatively small. Therefore a set-point change of 0.4 [rad] is given and the time signal measured. Figure H.2 shows the change in set-point (joint_setp/data) and the observed motion profiles (obs_state/data). The time it takes to reach the new set-pint is approximately 40[ms].

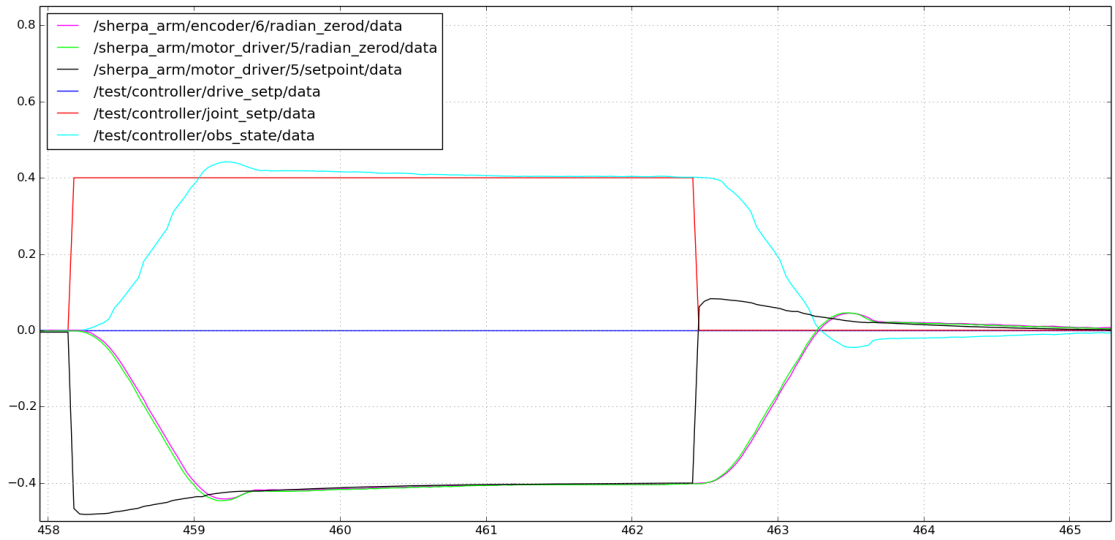


Figure H.2: Measurement set up to determine the latency in the image capturing process

H.2 Overview

The latencies of different parts of the system has been determined from the moment an image is captured until movement of the arm, assuming small displacements. The obtained latencies are presented in Table H.2. The total latency is determined at 190 milliseconds.

System	Average latency
(1) Logitech C920 HD Pro	±100 [ms]
(2) AR Track Alvar	±30 [ms]
(3) Pick place node processing	±20 [ms]
(4) Arm Controller	±40 [ms]

Table H.2: Latencies in the system

I Appendix: OptiTrack measurement setup

I.1 Measurements setup

For comparing the prediction results with the UAV its 'real time' position the data needs to be expressed in the same coordinate frame. The vision system provides information with respect to the camera frame and OptiTrack with respect to a predefined reference frame. By placing markers on the vision camera and the Parrot drone their positions can be measured with the OptiTrack system. The placement of these markers are depicted in Figure I.1.

(a) OptiTrack markers placed on the camera

(b) OptiTrack markers placed on a AR Parrot 2.0 Drone and gripper interface

Figure I.1: OptiTrack markers

Figure I.2 depicts a 3D plot of the different coordinate frames of the measurement setup as seen from the Optitrack system.

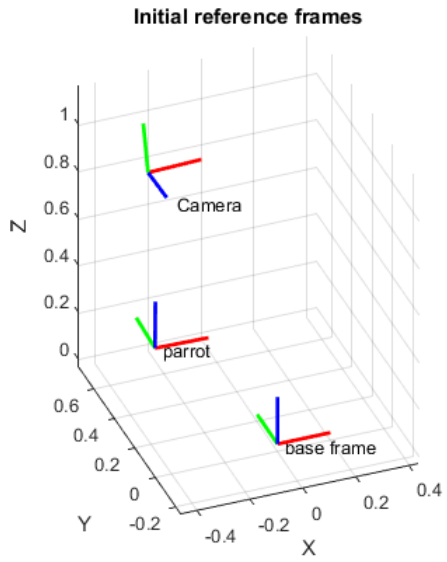


Figure I.2: Reference frames as seen from the Optitrack system with the vision camera facing downwards and drone positioned on the ground

By knowing the coordinates of the vision camera in the OptiTrack reference frame, the position of the AR Parrot drone as seen from OptiTrack can be transformed to the vision camera reference frame. The coordinate transforms are described by a homogeneous matrices $\mathbf{H}_b^a \in \mathbb{R}^{4x4}$, the superscript a defines the destination frame and subscript b defines the reference frame.

Equation I.1 describes the homogeneous matrix for a coordinate change from the OptiTrack Reference frame *OR*, to the OptiTrack Camera Reference *OCR*. The homogeneous matrix is composed of a rotation matrix $\mathbf{R}_{OR}^{OCR}$ and translation vector $\mathbf{t}_{OR}^{OCR}$.

$$\mathbf{H}_{OR}^{OCR} = \begin{bmatrix} \mathbf{R}_{OR}^{OCR} & \mathbf{t}_{OR}^{OCR} \\ 0 & 1 \end{bmatrix} \tag{I.1}$$

A second transformation is needed from the camera reference frame in OptiTrack to the cameras reference frame. Both reference frames are located on the camera, but differ in position and orientation, as shown in Figure I.3. The OptiTrack frames origin is actual placed as close to the cameras sensor as manually possible, however for clarity drawn on top of the camera. The transformation between the two frames is denoted as $\mathbf{H}_{OCR}^{CR}$. Initially this transformation only contains a rotations around the Z and X axis, assuming a similar origin of the two frames. Possible offsets due to miss alignment or inaccuracy can be accounted for by preforming a static measurement at forehand. Any offsets can be adjusted in the translation vector of the homogeneous matrix.

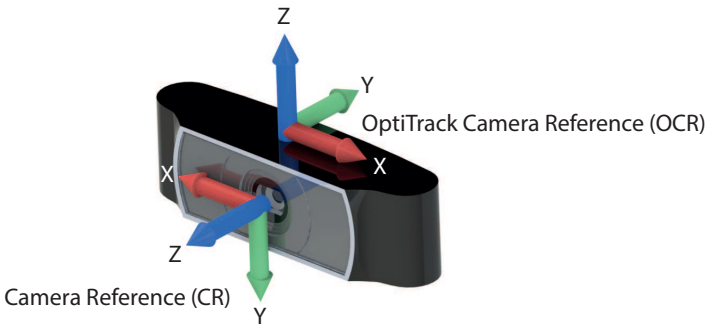


Figure I.3: Vision and OptiTrack reference frames on the camera, adapted from (TASKIRAN, 2016)

Multiplication of both coordinate transforms $\mathbf{H}_{OR}^{OCR}$ and $\mathbf{H}_{OCR}^{CR}$ results in a single homogeneous matrix from OptiTrack to the camera reference, denoted as $\mathbf{H}_{OR}^{CR}$. Using this coordinate transformation the UAV position in OptiTrack can be compared with the prediction results. Taking the inverse of $\mathbf{H}_{OR}^{CR}$ transforms coordinates defined with respect to the camera frame towards the OpiTrack reference frame and is denoted by $\mathbf{H}_{CR}^{OR}$.

The prediction results are measured with respect to the Tool Centre Point (TCP) of the robotic arm. The TCP is located at the centre of the gripper, as shown in Figure I.4. This reference allows for direct set-point usage for the control sequence of the arm, meaning no other transformations are required.

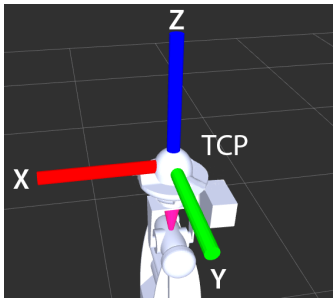


Figure I.4: TCP frame

For comparison a third transform in needed to transform the prediction results from the TCP frame towards the camera frame and is denoted by $\mathbf{H}_{TCP}^{VR}$. This is a fix transform and is directly available through the robot's kinematic chain in ROS. The total transform becomes $\mathbf{H}_{TCP}^{OR}$. To compute these transforms and compare the prediction results Matlab is used (MATLAB, 2016).

Bibliography

Alvarn, V. (2016), Vtt Alvar.
http://virtual.vtt.fi/virtual/proj2/multimedia/index.html

Andujar, C. (2017), Kinect.
http://www.cs.upc.edu/~virtual/RVA/CourseSlides/Kinect.pdf

APRIL, R. L. (2016), AprilTag.
https://april.eecs.umich.edu/

AslaTech (2017), AslaTech.
http://http://www.aslatech.com/

Baarsma, J. H. (2015), *Docking a UAV using a Robotic Arm and Computer Vision*, Msc report 008ram2015, University of Twente.

Barbieri, G. B. (2016), *Control architecture for docking UAVs with a 7-DOF manipulator*, Msc report 045ram2016, University of Twente.

Barrett, E., M. Reiling, G. Barbieri, M. Fumagalli and R. Carloni (2017), Mechatronic design of a variable stiffness robotic arm, pp. 4582–4588.

Barrett, E., M. Reiling, M. Fumagalli and R. Carloni (2016), The SHERPA gripper: Grasping of small-scale UAVs, in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 384–389, doi:10.1109/SSRR.2016.7784331.

Barrett, E., M. Reiling, S. Mirhassani, R. Meijering, J. Jager, N. Mimmo, F. Callegati, L. Marconi, R. Carloni and S. Stramigioli (2018), Autonomous Battery Exchange of UAVs with a Mobile Ground Base, in *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA), Submitted*, pp. 1–7.

Bohren, J. (2017), Smach.
http://wiki.ros.org/smach

Boterenbrood, W. (2015), Development of the Low-level Software Architecture for the Sherpa Robot Arm, Bsc report 024ram2015, University of Twente.

Cardiff.University (2013), MATLAB Digital Audio Effects.
http://www2.cmpe.boun.edu.tr/courses/cmpe362/spring2013/files/tutorial/FilteringExamples.pdf

Cnet (2007), Creative Live! Cam Optia AF Specifications.
https://www.cnet.com/products/creative-live-cam-optia-af/specs/

Cogniteam (2014), Decision making.
http://wiki.ros.org/decision_making

Conference, E. C. (1993), *European Control Conference 1993: Volume 2*, ECC.

DAQRI, A. C. L. (2017), ARToolKit.
https://archive.artoolkit.org/documentation/

Eitan Marder-Eppstein, V. P. (2007), ROS Action Library.
http://wiki.ros.org/actionlib

ELMO (2017), ELMO Motion Control.
http://www.elmomc.com/products/servo-drives.htm

Gassend, B. (2015), Rqt Dynamic Reconfigure.
http://wiki.ros.org/dynamic_reconfigure

Integrated Imaging Solutions Inc, F. (2017), Bumblebee2 1394a.
https://www.ptgrey.com/

bumblebee2-firewire-stereo-vision-camera-systems

Kohler, J., A. Pagani and D. Stricker (2010), Detection and Identification Techniques for Markers Used in Computer Vision, in *Visualization of Large and Unstructured Data Sets âĂŞ IRTG Workshop*, pp. 36–44, doi:10.4230/OASIcs.VLUDS.2010.36. http://vesta.informatik.rwth-aachen.de/opus/volltexte/2011/3095/pdf/6.pdf

Logitech (2017), Logitech C920 Specifications. http://support.logitech.com/nl_nl/product/hd-pro-webcam-c920/specs

Manaf A. Mahammed, AmeraI. Melhum, F. A. K. (2013), Object Distance Measurement by Stereo VISION, in *(2013) International Journal of Science and Applied Information Technology (IJSAIT)*, volume 2, pp. 05–08, ISSN 2278-3083.

Marconi, L., C. Melchiorri, M. Beetz, D. Pangercic, R. Siegwart, S. Leutenegger, R. Carloni, S. Stramigioli, H. Bruyninckx, P. Doherty, A. Kleiner, V. Lippiello, A. Finzi, B. Siciliano, A. Sala and N. Tomatis (2012), The SHERPA project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments, in *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–4, doi:10.1109/SSRR.2012.6523905.

MATLAB (2016), *version 9.1.0 (R2016b)*, The MathWorks Inc., Natick, Massachusetts.

Mooser, J., S. You and U. Neumann (2006), Tricodes: A Barcode-Like Fiducial Design for Augmented Reality Media, in *2006 IEEE International Conference on Multimedia and Expo*, pp. 1301–1304, doi:10.1109/ICME.2006.262777.

NaturalPoint (2017), OptiTrack Flex 3. http://optitrack.com/products/flex-3/

Niekum, S. (2017), AR Track Alvar. http://wiki.ros.org/ar_track_alvar

Nikolic, J., J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale and R. Siegwart (2014), A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM, in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 431–437, ISSN 1050-4729, doi:10.1109/ICRA.2014.6906892.

Pagliari, D., L. Pinto, M. Reguzzoni and L. Rossi (2016), INTEGRATION OF KINECT AND LOW-COST GNSS FOR OUTDOOR NAVIGATION, *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **vol. XLI-B5**, pp. 565–572, doi:10.5194/isprs-archives-XLI-B5-565-2016. https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B5/565/2016/

Patrick Mihelich, Kurt Konolige, J. L. (2017), Stereo Image Proc. http://http://wiki.ros.org/stereo_image_proc

Pitzer, B. (2016), ROS usb cam node. http://wiki.ros.org/usb_cam

QR.Generator (2014), QR Code Generator. http://nl.qr-code-generator.com/

Quigley, M., K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng (2009), ROS: an open-source Robot Operating System, in *ICRA Workshop on Open Source Software*.

RED.COM, L. (2017), GLOBAL and ROLLING SHUTTERS. http://www.red.com/learn/red-101/global-rolling-shutter

Sattar, J., E. Bourque, P. Giguere and G. Dudek (2007), Fourier tags: Smoothly degradable fiducial markers for use in human-robot interaction, in *Computer and Robot Vision, 2007*.

CRV '07. Fourth Canadian Conference on, pp. 165–174, doi:10.1109/CRV.2007.34.

Siltanen, S. (2012), *Theory and applications of marker-based augmented reality*, VTT Technical Research Centre of Finland.
http://www.vtt.fi/inf/pdf/science/2012/S3.pdf

Skybotix (2014), Visual-inertial Sensor factsheet.
http://www.skybotix.com/

Sucan, I. A. and S. Chitta (2017), MoveIt! Motion Planning Framework.
http://moveit.ros.org

Swanson, D. C. (2011), *Signal Processing for Intelligent Sensor Systems with MATLABÂő, Second Edition*, CRC Press.

Tan, D. J., D. M. Brouwer, M. Fumagalli and R. Carloni (2017), A 2-DOF Joint With Coupled Variable Output Stiffness, pp. 366–372, doi:10.1109/LRA.2016.2631730.

TASKIRAN, A. (2016), Logitech C920 Webcam Render.
https://grabcad.com/library/logitech-c920-webcam-1

Werink, R. (2016), Design and realization of a gripper for the SHERPA robotic arm, Bsc report 013ram2016, University of Twente.

Willow Garage, R. c. (2013), ORK: (O)bject (R)ecognition (K)itchen.
https://github.com/wg-perception/object_recognition_core