

AUTONOMOUS FRUIT PICKING WITH A TEAM OF AERIAL
MANIPULATORS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TAHSİNCAN KÖSE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

**AUTONOMOUS FRUIT PICKING WITH A TEAM OF AERIAL
MANIPULATORS**

submitted by **TAHSİNCAN KÖSE** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Şeyda Ertekin
Supervisor, **Computer Engineering, METU**

Examining Committee Members:

Assist. Prof. Dr. Hande Alemdar
Computer Engineering, METU

Assoc. Prof. Dr. Şeyda Ertekin
Computer Engineering, METU

Assoc. Prof. Dr. Ahmet Murat Özbayoğlu
Computer Engineering, TOBB UET

Date: 07.09.2021



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Tahsincan Köse

Signature :

ABSTRACT

AUTONOMOUS FRUIT PICKING WITH A TEAM OF AERIAL MANIPULATORS

Köse, Tahsincan

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Şeyda Ertekin

September 2021, 103 pages

Manipulation is the ultimate capability for autonomous micro unmanned aerial vehicles (MAVs), which would enable a substantial number of novel use-cases. Precision agriculture is such a domain with plenty of practical problems that could utilize aerial manipulation, which is faster with respect to ground manipulation. Apple harvesting is the most prominent use case with ever-growing percentages in the overall apple production costs due to increasing imbalance between labor supply and demand. Moreover, contemporary industrial UGVs equipped with arms still strive to compete against human labor financially, which is not the case for relatively cheaper aerial manipulators. Nevertheless, the dynamics of the aerial manipulator is significantly more convoluted when compared to plain quadrotors and thus demand a more sophisticated control pipeline. In this thesis, a brief dynamics analysis on the airframe with a 3 DOF manipulator is conducted and subsequently an appropriate autopilot software with an adequately stable control pipeline -PX4- is selected. Thereafter RRT path planning algorithm family, followed by its multi-robot variant sRRT, which uses subdimensional expansion framework, is introduced. Then the inverse kinematics of 3 DOF

arm is introduced, which is used alongside with the local¹ planner in fruit picking. Consecutively a supervised apple detection model based on Mask R-CNN and Data Distillation with real-time capable inference is developed. At the end, multi-robot apple picking scenario is realized and the integration of these submodules is verified in simulation.

Keywords: Robotics, Aerial Manipulation, Precision Agriculture, Fruit Recognition, Multi-robot Path Planning, sRRT, Mask R-CNN



¹ the word “local” is used to denote the particular planner of a single robot, not in the sense of local planning.

ÖZ

HAVA MANİPÜLATÖRÜ TAKIMIYLA OTONOM MEYVE TOPLAMA

Köse, Tahsincan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Şeyda Ertekin

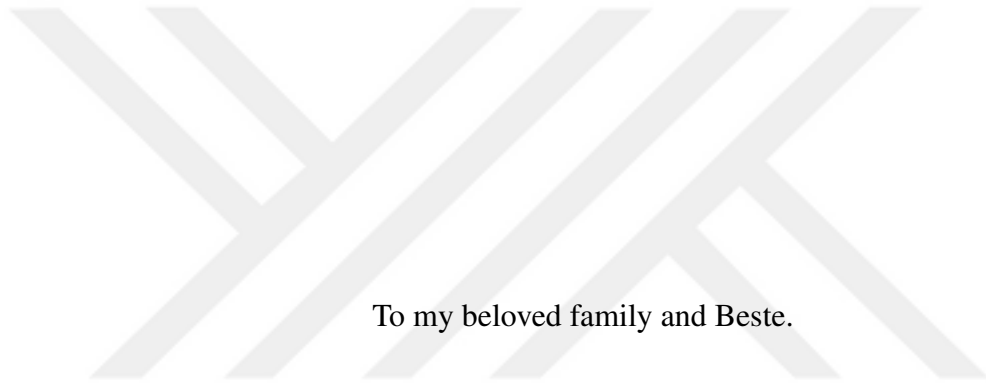
Eylül 2021 , 103 sayfa

Otonom mikro İHAların nihai kabiliyetleri etraflarını manipüle etme yeteneğidir. Bu yetenek bir sürü yeni kullanım senaryosuna fırsat verecektir. Bu tip senaryolardan çok sayıda problem barındıran hassas tarım, yer manipülasyonundan daha hızlı olan hava manipülasyonundan faydalanabilir. Giderek derinleşen emek arz-talep dengesizliğinden dolayı toplam elma üretimi giderlerinde sürekli büyüyen bir yüzdeye sahip olan elma hasadı bu senaryoların en meşhurdur. Buna ek olarak, manipülatörlerle teçhiz edilmiş çağdaş endüstriyel İKAlar hala insan emeği ile finansal olarak rekabet etmeye çalışmaktadır. Daha ucuz olan hava manipülatörleri ise maliyet konusunda karadaki rakiplerinden öndedirler. Bununla birlikte, hava manipülatörlerinin dinamik profilleri simetrik dört-kanatlılara göre çok daha karmaşıktır, dolayısıyla daha sofistike bir kontrol şemasına ihtiyaç duyarlar. Bu tezde, 3 özgürlük derecesine sahip bir kol takılmış dört-kanatlı bir İHA'nın kısa dinamik analizi yapılmış ve devamında bu dinamik profiline uygun kontrol şemasına sahip bir otopilot olan PX4 seçilmiştir. Devamında, RRT yol planlama algoritma ailesi ve onun altboyutsal genişleme yöntemini kullanan çoklu-robot türevi sRRT tanıtılmıştır. Sonra, meyve toplama görevinde son etapta kullanılan yerel yol planlacısıyla birlikte çalışan robotik kolun ters kinematiği iş-

lenmiştir. Buna müteakip, elma tanıma kabiliyeti bulunan, Mask R-CNN mimarisini ve Veri Damıtma metodunu kullanan bir gözetimli öğrenme modeli geliştirilmiştir. Son olarak çoklu-robot elma toplama senaryosu ve bu alt-modüllerin entegrasyonu simülasyon ortamında gerçekleştirilmiştir.

Anahtar Kelimeler: Robotik, Hava Manipülasyonu, Hassas Tarım, Meyve Tanıma, Çok Robotlu Yol Planlama, sRRT, Mask R-CNN





To my beloved family and Beste.

ACKNOWLEDGMENTS

I wish to express my deepest gratitude to my supervisor, Assoc. Prof. Dr. Şeyda Ertekin, who provided me the perfect balance of freedom and responsibility during the creation of this thesis. She responded to me every time when I needed her supervision, even at late night meetings lasting for hours. It would not possible to finish this thesis without her practical attitude that pulled me back onto the track each time I started to get lost in the details and attempted sideways that were little to no help in the scope of this thesis. Also her elaborate review process on the delivered results maintained the quality of this thesis from the beginning to the end. I would like to continue with my beloved girlfriend, Beste, who beared me on getting stuck in front of computer for countless days and the sudden early holiday returns I had to make for this thesis. You are my angel.

I wish to acknowledge the support and great love of my family -my mother, father, brother and grandmother- whose sole existence refills my motivation to the limits. Finally I want to sincerely thank to my friends and colleagues from team Collaborator; Raşit, Burak and Cansın, who possess both the most brilliant sense of humor and insightful ideas that kept me socially and professionally awake.

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vii
ACKNOWLEDGMENTS	x
TABLE OF CONTENTS	xi
LIST OF TABLES	xv
LIST OF FIGURES	xvi
LIST OF ABBREVIATIONS	xix
LIST OF SYMBOLS	xx

CHAPTERS

1 INTRODUCTION	1
1.1 Problem Statement	2
1.2 Objectives	4
1.3 ROS	5
1.4 Gazebo	5
2 DYNAMICS AND CONTROL OF THE AERIAL MANIPULATOR	9
2.1 Controlling an Aerial Manipulator with PX4 in Gazebo	12
2.1.1 Position Control	16

2.1.2	Control at Hover conditions in Gazebo	18
2.1.3	Takeoff	20
2.1.4	Full Rotation	21
2.1.5	Idle	21
2.1.6	Under Manipulation Effect	21
3	PLANNING	27
3.1	Single UAV Planning	29
3.1.1	RRT	29
3.1.2	RRT*	30
3.1.3	Bidirectional RRT*	31
3.1.4	Bidirectional RRT* with Heuristic	31
3.1.5	Implementation	32
3.1.6	Evaluation - Metrics	32
3.2	Multi-UAV Planning	36
3.2.1	Sub-dimensional Expansion	36
3.2.2	sBiRRT*	36
3.3	Manipulator Planning	47
4	FRUIT DETECTION	51
4.1	Related Work	53
4.1.1	ResNet	53
4.1.2	R-CNN	55
4.1.3	Fast R-CNN	56
4.1.4	Faster R-CNN	59

4.1.5	Mask R-CNN	62
4.1.6	FPN	63
4.1.7	Apple Detection	64
4.2	Method	65
4.2.1	Dataset Preparation	66
4.2.2	Training	69
4.2.3	Speed vs. Accuracy Trade-off	72
4.2.4	Data Distillation	73
5	SOFTWARE ARCHITECTURE	79
5.1	Simulation	81
5.2	PX4 Parameters	81
5.3	TF	81
5.4	QuadManipulator	85
5.4.1	Motion Primitives	85
5.4.1.1	Offboard Position Control	86
5.4.1.2	Takeoff	86
5.4.1.3	Landing	86
5.4.1.4	Go to Home	86
5.4.1.5	Orbit with Finite State Machine	87
5.4.1.6	Plan Arm	88
5.4.1.7	Move Arm	88
5.4.1.8	Grasp & Dispose	89
5.4.2	Recognition	89

5.4.2.1	OpenCV Baseline	89
5.4.2.2	Fast DeepApple	91
6	CONCLUSION	93
	REFERENCES	97



LIST OF TABLES

TABLES

Table 3.1	Common RRT Hyperparameters	34
Table 3.2	RRTs Success Rate	35
Table 4.1	AP, F1 and FPS scores of Mask R-CNN on Apple detection	69
Table 4.2	Trade-off between accuracy and speed of different parameter combinations	72
Table 4.3	Accuracy scores after Data Distillation	76
Table 5.1	PX4 Parameters	82
Table 5.2	Accuracy of OpenCV baseline detector with different orbit behaviors	91

LIST OF FIGURES

FIGURES

Figure 1.1	Orchard with randomly planted trees - orthogonal view	5
Figure 1.2	Apple orchard in Gazebo	6
Figure 2.1	Aerial Manipulator	10
Figure 2.2	Full dynamics of Aerial Manipulator under the effect of J_1 , J_2 and J_3 motions	11
Figure 2.3	[PX4 and Gazebo on SITL], (n.d.), <i>ROS with Gazebo Simula- tion</i> , https://docs.px4.io/master/en/simulation/ros_ interface.html	14
Figure 2.4	Medium-sized Orchard Patrol Pattern	15
Figure 2.5	[Multicopter Control Architecture], (n.d.), <i>Controller Diagrams</i> , https://docs.px4.io/master/en/flight_stack/controller_ diagrams.html , PX4 Control Architecture	16
Figure 2.6	PX4 Translational Controllers	16
Figure 2.7	[Multicopter Attitude Controller], (n.d.), <i>Controller Diagrams</i> , https://docs.px4.io/master/en/flight_stack/controller_ diagrams.html	17
Figure 2.8	[Multicopter Angular Rate Controller], (n.d.), <i>Controller Dia- grams</i> , https://docs.px4.io/master/en/flight_stack/ controller_diagrams.html	18

Figure 2.9	Takeoff behavior with different noise models in Rviz	20
Figure 2.10	Robot responses to 2π rotation around z_B with different noise models	22
Figure 2.11	Idle with Default noise model	23
Figure 2.12	Idle with Zero noise model	23
Figure 2.13	Robot responses to manipulator motions with different noise models	24
Figure 3.1	RRT Algorithms	33
Figure 3.2	RRT Algorithms - Continued	33
Figure 3.3	RRT Metrics	34
Figure 3.4	Initial state after retrieving policy tree for each robot	40
Figure 3.5	sBiRRT* robot-robot collision in Step-1	41
Figure 3.6	sBiRRT* robot-robot collision resolution in Step-2	42
Figure 3.7	sBiRRT* robot-robot collision in Step-3	43
Figure 3.8	sBiRRT* robot-robot collision resolution in Step-4	44
Figure 3.9	sBiRRT* robot-robot collision in Step-5	45
Figure 3.10	sBiRRT* robot-robot collision resolution in final step	46
Figure 3.11	Arm Motion with IK Solution	49
Figure 4.1	Residual learning: a building block. Reprinted from <i>Deep Residual Learning for Image Recognition</i> by Kaiming He et al. 2016, CVPR .	54
Figure 4.2	Bottleneck building block	54
Figure 4.3	R-CNN Architecture	55

Figure 4.4	Fast R-CNN Architecture	57
Figure 4.5	Faster R-CNN Architecture	60
Figure 4.6	Region Proposal Network	62
Figure 4.7	FPN's Pyramid of Features	64
Figure 4.8	Mask R-CNN Architecture	65
Figure 4.9	Filtered Annotation Example	67
Figure 4.10	AP scores of the models trained with filtered and unfiltered annotations	68
Figure 4.11	Hyperparameters vs. Validation Loss	70
Figure 4.12	Training vs Validation Losses	71
Figure 4.13	Apple detections on a test-set image	71
Figure 4.14	Combined view of Teacher and Student models in Data Distillation setting	77
Figure 5.1	Overall System Architecture	80
Figure 5.2	Overall System Finite State Machine	80
Figure 5.3	Transformation Tree with PX4 NED Frame	83
Figure 5.4	Unified Transformation Tree with Multiple Robots	85
Figure 5.5	Orbit with Finite State Machine	87

LIST OF ABBREVIATIONS

ROS: Robot Operating System
IPC: Inter-process Communication
UAV: Unmanned Aerial Vehicle
UGV: Unmanned Ground Vehicle
MAV: Micro Unmanned Aerial Vehicle
VTOL: Vertical Take-off/Landing
DOF: Degrees of Freedoms
CoM: Center of Mass
PID: Proportional Integral Derivative
GCS: Ground Control Station
HITL: Hardware-in-the-Loop
SITL: Software-in-the-Loop
API: Application Programming Interface
RRT: Rapidly Exploring Random Tree
BiRRT: Bidirectional Rapidly Exploring Random Tree
CNN: Convolutional Neural Network
R-CNN: Region-based CNN
RoI: Region of Interest
SVM: Support Vector Machines
RPN: Region Proposal Network
IoU: Intersection over Union
FPN: Feature Pyramid Network

LIST OF SYMBOLS

Σ_A : Coordinate frame A

$^A p$: Point expressed in frame A

$^A t_B$: Translation vector of frame B with respect to frame A

$^A R_B$: Rotation matrix of frame B with respect to frame A

$^A T_B$: Homogeneous transformation matrix of frame B with respect to frame A

$^A \tau$: Torque affecting on frame A .

T : Thrust.

x^* : A desired state, e.g. desired position. Usually used in controller and planning domains.

CHAPTER 1

INTRODUCTION

An Unmanned Aerial Vehicle (UAV) is an aircraft controlled either autonomously, semi-autonomously, or remotely. The fundamental principle in all three modes of operation is the absence of human operators on board. Numerous types and sizes of UAVs have been introduced in academia and industry until recently, which has resulted in a great variety of vehicles. Due to the large number of different types of vehicles, specific classification metrics have been introduced to categorize them concerning particular aspects and use cases. Some such criteria are weight, size and wing type. In the writing of this thesis, a Vertical Take-off/Landing (VTOL) quadcopter¹ is used, which is probably the most studied type of vehicle in academic contexts.

The recent decade witnessed significant technological advancements in Robotics, which comprises the burst of quadcopters in many different use-cases, e.g. surveillance, industrial inspection and crash site reconnaissance. In most of them though, the vehicle cannot manipulate the environment due to a lack of suitable actuators, which prevents the possibility to scale for much more advanced use-cases. **Aerial manipulation** is a recently emanated field that focuses on the design, integration, control and motion planning of such actuators –typically robotic arms– along with the aircrafts so as to enable the vehicles to actively manipulate, e.g. grasp and pick & place, the environment they are flying in. Due to the convoluted nature of dynamics involved in this type of vehicle, it had been considered very difficult to develop stable models that fully describe the physics[1]. Therefore early studies in the field embrace a *simplified approach*, in which the masses of manipulators and target entities were much smaller with respect to the UAVs, which allowed the researchers to ignore the changing center

¹ a multi-copter with four rotors

of gravity (CoG) and inertia during a manipulation session[2] and use the native flight controllers. Nevertheless such an assumption rapidly becomes invalid as the size and mass of a rotary system diminish, which was another hot topic of the last decade that led to the invention of numerous miniature and battery-optimized UAVs. To that end, various systematic control methods have been developed to solve the problem from different aspects[3].

Aerial manipulation unveils its true value in use-cases *such as assembly or contact inspection, in locations that cannot be accessed or that are very dangerous or costly to be accessed from the ground*[4]. All in all, it requires the highest level of autonomy – autonomous physical interactions[4], which pushes the boundaries from the non-intrusive behaviors, e.g. *sense and avoid, search and track and target detection*; to the intrusive ones, e.g. *cargo delivery, fruit picking and load transportation*.

Agricultural robotics is one such sub-domain that requires autonomous physical interactions. In this context, UGVs equipped with manipulators make reaching hard surfaces and hollows possible with the robotic arms' added Degrees of Freedoms(DOF). Accordingly, labor-intensive tasks such as fruit picking[5], weed control[6] and harvesting[7] are typical yet critical instances of agricultural problems already solved in carefully constructed scenarios with ad-hoc methods. Nevertheless, more robust solutions are still being actively sought through particular funding programs and competitions in the scope of H2020 such as Agri-food Competition for Robot Evaluation(ACRE)[8] as these types of problems remain to be solved from the general perspective.

1.1 Problem Statement

With the outbreak of the COVID-19 pandemic, the need for advanced automation capabilities in agriculture became more evident than ever before as the human labor shortage, which was already existing in pre-pandemic conditions, correlatively severed. In the early stages of the outbreak, a set of social distancing rules were

implemented around the globe in order to reduce direct human interaction. Indirect human interaction is similarly considered effective in the spread of the disease by the governmental authorities. However, a very recent not peer-reviewed study states that there is no explicit evidence on the dominance of indirect exposure [9]. The food supply chain is one such area where humans are still playing a crucial role at all levels. Although *European Commission Directorate-General for Health and Food safety* denotes in its **Food hygiene** report[10] that the infection via food consumption is not recorded to date, food packaging may still be susceptible to the viral transmission, thus must be conducted conforming to good hygiene practices. It is consequently apparent that the minimization of human agents in the food supply chain is an inhibiting factor in the spread of disease.

Another important aspect of agricultural automation is its dramatic efficacy compared to human labor, which is already severed as described above. The world society needs more structured, sophisticated and environment-friendly practices in agriculture to combat the major upcoming problems such as global warming and food shortage due to the ever-increasing population. Therefore autonomous systems that would be able to work with minimal human intervention are essential to stable and sufficient mass production & delivery of food supplies. Considering all these factors, the need for agricultural automation would persist, let alone not increase, in the post-pandemic world.

Aerial manipulators with their $4+n - n$ depend on the number of joints of the attached manipulator— DOF possess a lot more potential than their UGV competent due to their inherently superior manipulation capabilities and speed. These allow them to:

- i) access otherwise unreachable surfaces and hollows,
- ii) cover a much broader area in a time unit.

Moreover, these types of autonomous agents could be *cooperatively* used with the other robots, e.g. UGVs and monitor-only UAVs, using information/belief sharing techniques[11] to fulfill the agricultural tasks more efficiently.

1.2 Objectives

Aligned with the problem statement, this thesis will attack one such agricultural use-case wherein the minimum inevitable human intervention - monitoring - abides. Robotics, in general, is a vast field that is inconvenient to solve whose all problems. Thus this thesis specializes in certain aspects of the problem mentioned above.

The task of this thesis is **not** to provide a low-level control pipeline for the aerial manipulator to stably navigate and interact within the environment, thus it nor:

- i) designs and implements a control pipeline,
- ii) provides an extensive analysis on how a control policy can be applied to the aerial manipulator.

It instead uses a ready-made model from the literature[12] by providing a brief analysis of the disturbances introduced by the manipulator on the extended model. Thereafter it provides a non-exhaustive list of the studies in the literature on the control algorithms that intelligently compensate for these disturbances and allow real robot execution. Any one of such control methods is assumed to be already *implemented* on the model and perform sufficiently accurate in this study so as to allow focusing on design, implementation and integration of a software stack, which would do:

- i) the task-level planning of multiple aerial manipulators,
- ii) the efficient motion planning of aerial manipulator, addressing the whole-body problem (i.e. motion planning of both UAV and manipulator),
- iii) real-time object² recognition with the attached camera to the aerial frame, using a supervised machine learning model.

The sample environment is an orchard containing apple trees in Gazebo simulator[13]. Since multiple UAVs are involved in the study, it is intractable to physically verify the software stack in its entirety. The outcome of this thesis will be the successful demonstration of an autonomous pick & place scenario in an apple orchard with multiple

² specifically fruit

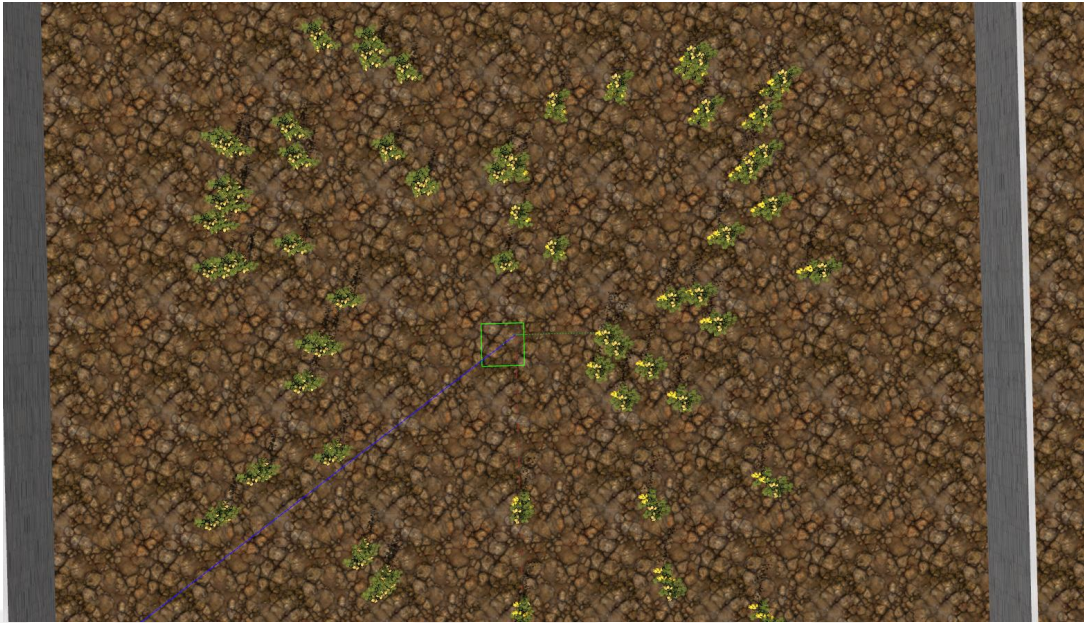


Figure 1.1: Orchard with randomly planted trees - orthogonal view

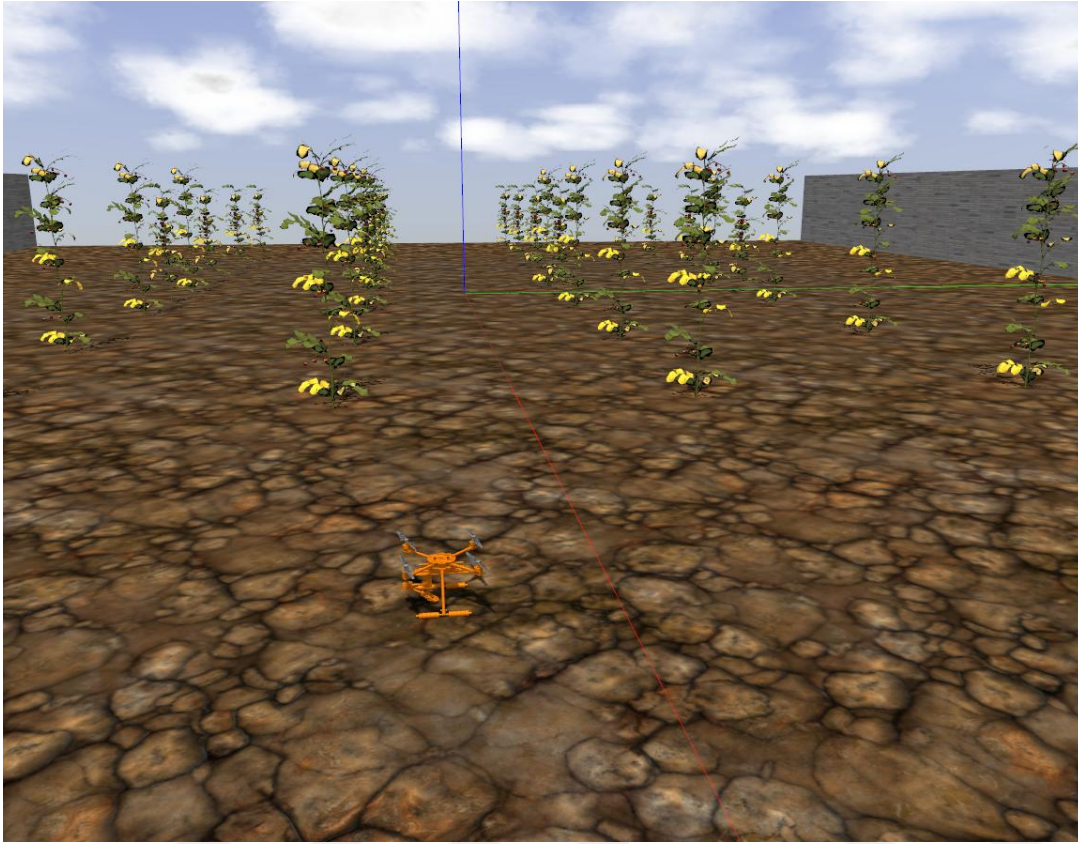
aerial manipulators in simulation in a time-efficient manner.

1.3 ROS

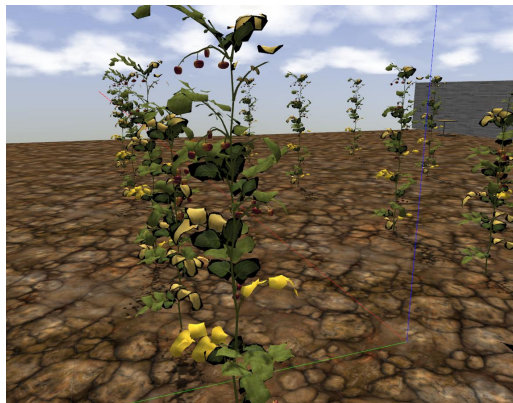
ROS is a robot-agnostic middleware with a collection of libraries and tools to facilitate robotics software development. It leverages the connected processes as ROS nodes and then its advanced IPC capabilities -topics, services, actions- become readily available to them. It exposes plugin-based architectures in most layers, rendering itself quite flexible and open to integration to all sorts of different software stacks. PX4 is one such autopilot software stack that has native ROS integration.

1.4 Gazebo

Gazebo is unofficially the flagship physics simulator of ROS community, which has by far the most development effort and hence matured tools and plugins. It has a similar level of flexibility as ROS; therefore it is significantly easy to introduce new models, robots, sensors, behaviors and so on. Bidirectional communication channels



(a) Orchard with regularly planted trees



(b) Apple plants

Figure 1.2: Apple orchard in Gazebo

of simulation process is exposed as a ROS node ([gazebo_ros](#)) just as the other processes.





CHAPTER 2

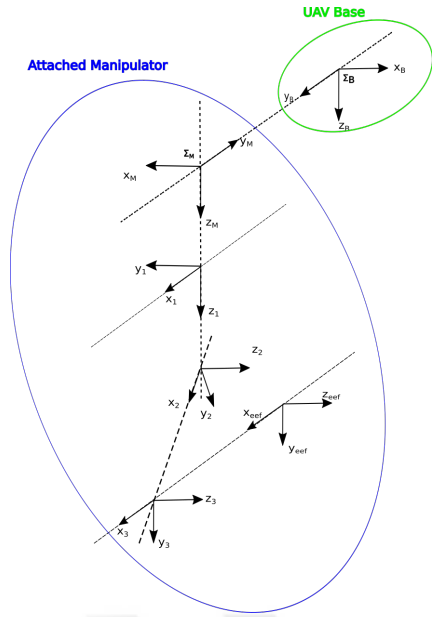
DYNAMICS AND CONTROL OF THE AERIAL MANIPULATOR

There are two main approaches on identifying the strong dynamic coupling between UAV and manipulator:

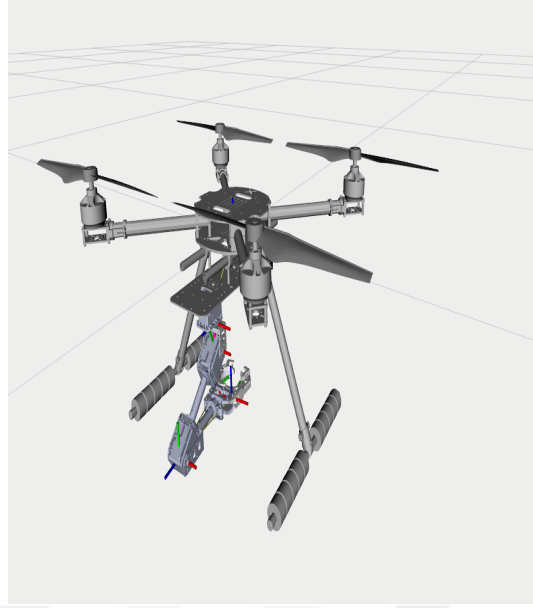
1. By using interaction force/torques between subsystems.
2. By using variable inertial parameters, i.e. center of mass (CoM) and inertia matrix.

The chosen identification approach significantly affects the control algorithm. For instance, the former approach dictates the usage of disturbance observers — the components utilized in the computation of compensating inputs for the respective control algorithm — in both subsystems. On the contrary, the latter identification method leads to a unified control algorithm that considers the entire aerial manipulator as a single controllable object. The type of application could simplify the complexity of the control problem. **Hovering manipulation tasks** are such applications, which is also the use-case of interest of this thesis, where the manipulator end of the overall control system ignores the coupling effect of the UAV since the UAV will not make any translational or rotational motion during arm manipulation, hence uses a plain manipulation control law. Contrarily, UAV dynamic modeling and control are performed under disturbances introduced by the manipulator.

Plain quadrotor dynamics has become a classical textbook context, making it redundant to reiterate here. Having said this, a concise yet sufficient analysis can be found in Section 4.2 of [14] by the aspiring reader. Contrarily aerial manipulator dynamics are much more complicated than the plain one since now at least one manipulator with multiple links and joints is in the equation. Moreover, the symmetry in the dynamics



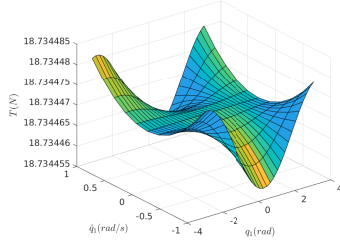
(a) Kinematic Tree



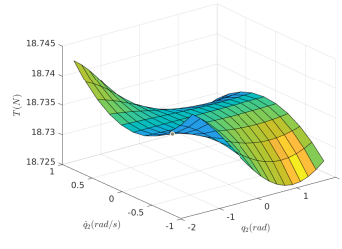
(b) Model

Figure 2.1: Aerial Manipulator

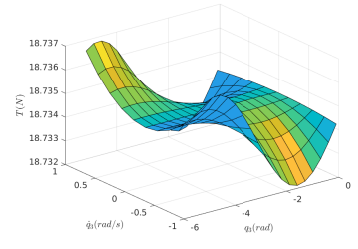
is no longer sustained as the relative poses of the links of the manipulator are subject to change, which leads to variable CoM and thus inertia matrix. The kinematic tree (seen in Figure 2.1a) of the aerial manipulator, along with its dynamic structure -mass, inertia matrix and CoG of the constituent links- need to be piped into Recursive Newton-Euler algorithm [15][16][4] so as to compute the particular force/torque effects of joints on Σ_B .



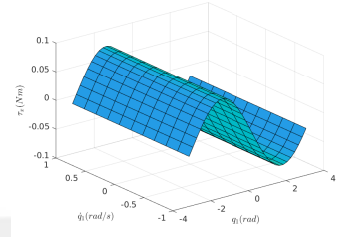
(a) T due to J_1



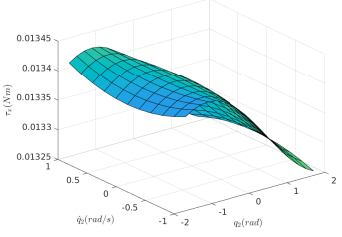
(b) T due to J_2



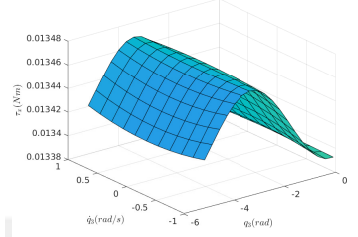
(c) T due to J_3



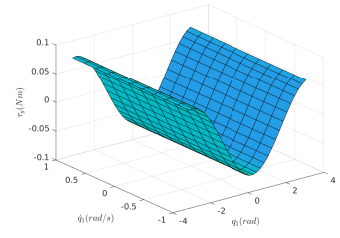
(d) ${}^B\tau_x$ due to J_1



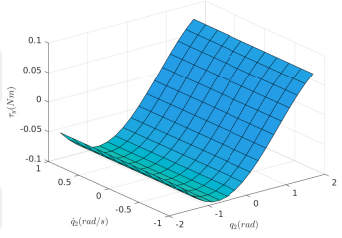
(e) ${}^B\tau_x$ due to J_2



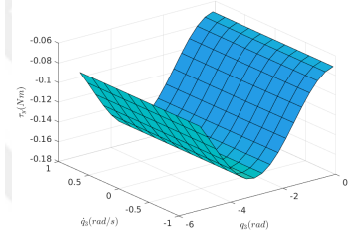
(f) ${}^B\tau_x$ due to J_3



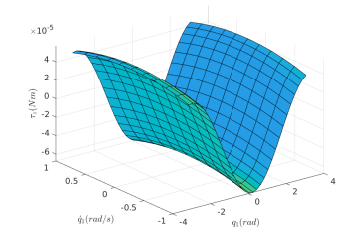
(g) ${}^B\tau_y$ due to J_1



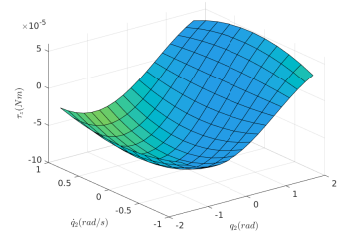
(h) ${}^B\tau_y$ due to J_2



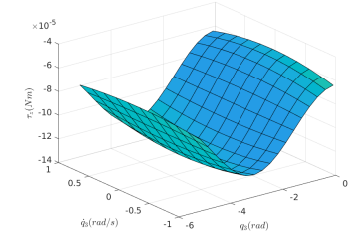
(i) ${}^B\tau_y$ due to J_3



(j) ${}^B\tau_z$ due to J_1



(k) ${}^B\tau_z$ due to J_2



(l) ${}^B\tau_z$ due to J_3

Figure 2.2: Full dynamics of Aerial Manipulator under the effect of J_1 , J_2 and J_3 motions

Figure 2.2 sketches an extensive picture of the aerial manipulator dynamics, when the joints happen to be in certain configurations within their respective joint limit ranges

with varying instantaneous velocities, which constitute a grid of inputs. The UAV is assumed to be in the hovering state all the time and other joints are assumed to be in 0-configuration. Also, all joints have zero instantaneous acceleration, which is a fair assumption for the particular robot at hand. Both T and ${}^B\tau$ profiles of J_1 follow a symmetric manifold. However, it is not the case for the subsequent joints, when the coupled dynamics start to affect UAV body due to physical structure.

Joint velocities have secondary importance according to plots in Figure 2.2. Even further, they seem to have no effect on J_1 , which is a facilitating factor for the controller pipeline. Despite not being entirely negligible as in the case for J_1 , other joints also do not suffer from any adverse effects of the velocities. On the other hand, certain configurations of J_2 and J_3 demand more moments to be generated around x and y axes of Σ_B , which is the task of any capable controller we have assumed since the beginning. Finally, even a simple PID controller, which lacks such capabilities, might perform roughly reasonable as long as:

- the configuration of the joints are kept strictly in the subsurface around the global minimum of plots in Figure 2.2,
- joints are moved one after the other.

Having said these, we finalize the dynamics analysis of the aerial manipulator and proceed to control in Gazebo with PX4 autopilot stack.

2.1 Controlling an Aerial Manipulator with PX4 in Gazebo

PX4[17] is the most advanced and capable open-source robotics autopilot framework for various robots such as aerial robots, autonomous submarines and ground vehicles. It provides easily extendable interfaces for custom software and hardware integration. Therefore, it is the most used open-source tech stack in industry and academia. Furthermore, it has native ROS and Gazebo support, aside from a number of other simulators, which makes it the best candidate for this thesis.

PX4 uses `mavros`[18] node to communicate with Gazebo over MAVLink[19], a protocol that is particularly designed for bi-directional communication between micro aerial vehicles¹ and ground control stations (GCSs). Communication with Gazebo happens through a set of Gazebo plugins implemented by PX4 community, which subscribe to actuator topics and publish to sensor topics provided by `mavros` node. Not surprisingly, PX4 publishes to actuator topics and subscribes to sensor topics provided by `mavros` node. `mavros` is the standardization layer between PX4 and Gazebo, which allows the autopilot stack to disregard whether the received data comes from a simulated, emulated or real world. It simply receives raw sensor readings and ultimately computes actuation commands in order to publish to the topics provided by `mavros` node. Such an encapsulation in communication caters three different modes of operation:

- **Hardware-in-the-loop (HITL):** The flight code/application runs on the target hardware, typically connected through a serial comm. slot to a companion computer that runs the simulation software. Sensor readings still come from the simulation; however, PX4 generates actuation commands on the target hardware, which is closer to reality and more preferable in late dry-run tests from certain aspects.
- **Software-in-the-loop (SITL):** The flight code/application runs on the developer computer. Serial communication is simulated via TCP ports. SITL is the mode of operation that is used in this thesis.
- **Real-world execution:** The flight code/application runs on target hardware connected to the lower-level controller board. Actuation commands are generated in target hardware and input to the controller board, which employs internal computations to send electrical signals to motors/ESCs.

¹ also supports MAV-to-MAV communication.

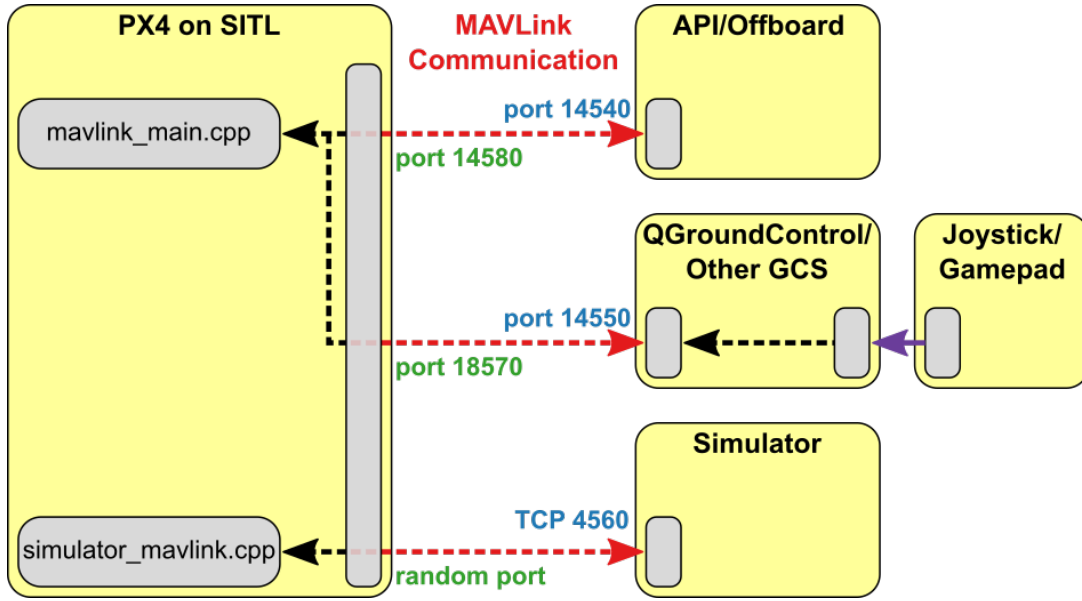


Figure 2.3: [PX4 and Gazebo on SITL], (n.d.), *ROS with Gazebo Simulation*, https://docs.px4.io/master/en/simulation/ros_interface.html

Figure 2.3 represents a broad picture of SITL mode. The dashed arrows are the messages passed over the ROS topics, subscribed and published by the corresponding ROS nodes (processes). There are two main ways of sending high-level commands to PX4 autopilot, offboard API and GCS². Offboard API is generally used by applications that involve more autonomy, whereas GCSs provide comprehensive GUIs to perform semi-autonomous tasks. Nevertheless, these two don't exclude each other from the same run-time and can be used alongside.

In a nutshell, this work implements a high level multi-UAV global planning component to produce environment-agnostic motion primitives (patterns), which are used to cover a wide range of areas; whereas the local planning component computes agile maneuvers for evasive motions from obstacles/hazards. As expected, local planning takes control of the robot, evades the immediate collision by executing local plans and consequently passes control back to a globally planned trajectory by joining it at a convenient viapoint.

² `QGroundControl` is used in this thesis.

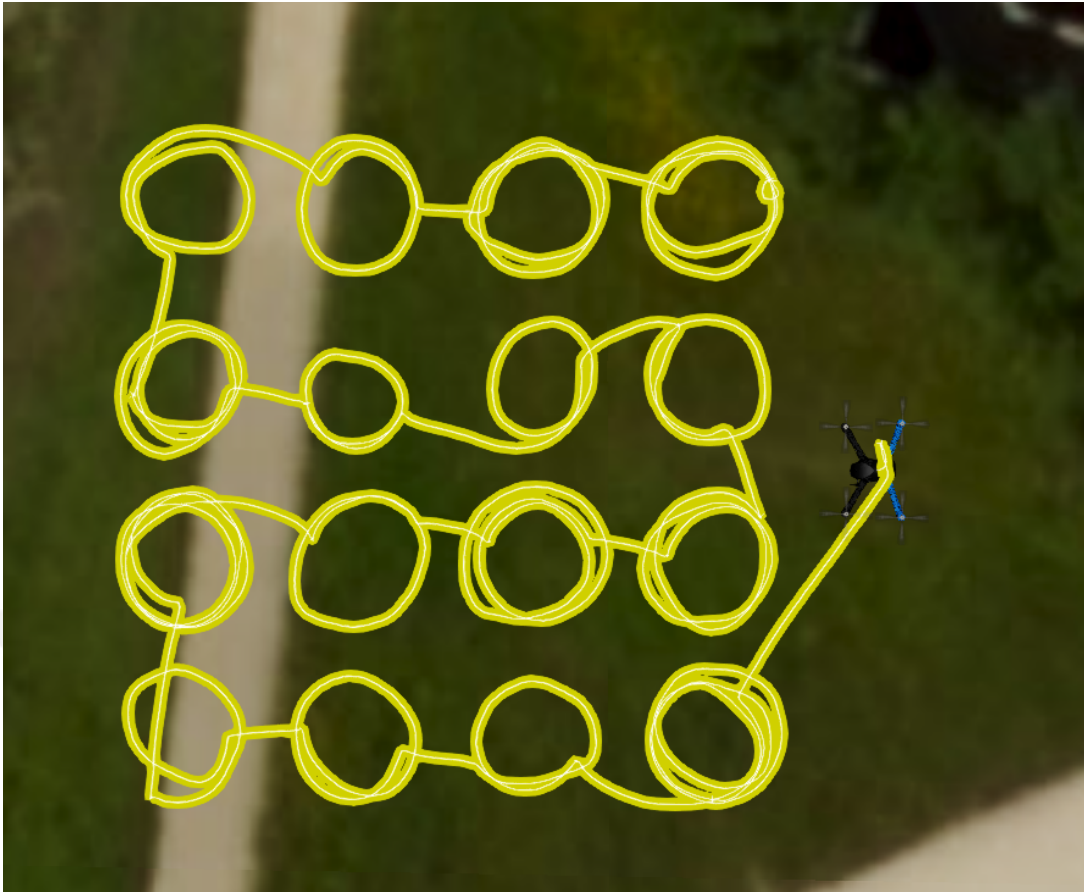


Figure 2.4: Medium-sized Orchard Patrol Pattern

Figure 2.4 contains a global collision-unaware path tracked with an assortment of orbit and offboard position control motion primitives. Lower level controllers of the aerial manipulator³ that actualize these motion primitives are briefly explained in the following sections.

³ actually they belong to PX4 autopilot

2.1.1 Position Control

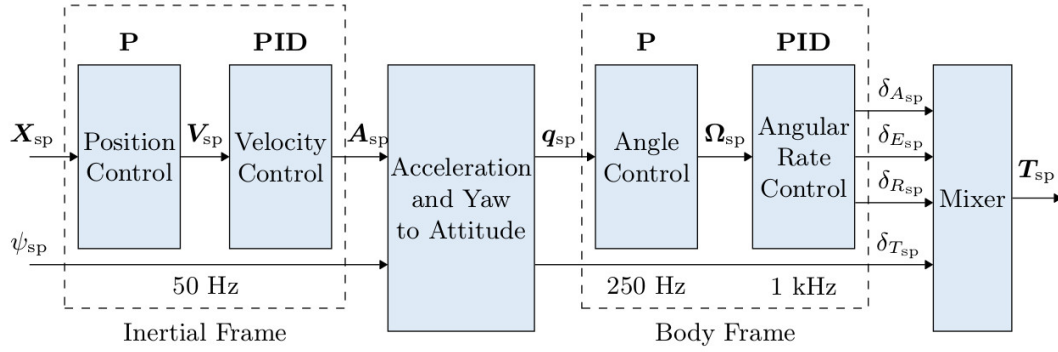
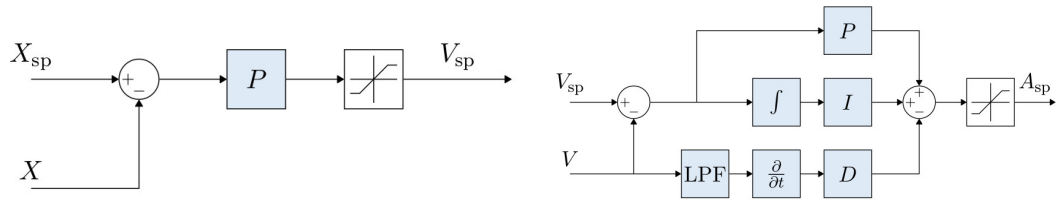


Figure 2.5: [Multicopter Control Architecture], (n.d.), *Controller Diagrams*, https://docs.px4.io/master/en/flight_stack/controller_diagrams.html, **PX4 Control Architecture**

In Figure 2.5, the overall architecture is visible. It is actually a standard cascaded PID control scheme, where the constituent controllers employ a mix of P, PID and K-PID mechanisms.⁴



(a) [Multicopter Position Controller], (b) [Multicopter Velocity Controller],
(n.d.), *Controller Diagrams*, https://docs.px4.io/master/en/flight_stack/controller_diagrams.html (n.d.), *Controller Diagrams*, https://docs.px4.io/master/en/flight_stack/controller_diagrams.html

Figure 2.6: PX4 Translational Controllers

⁴ The notation used in PX4 is slightly different than the one used in Sec. 4.2 of [14] and some terms don't have linear mapping. $()_{sp}$ denotes the target/goal of the component we are either directly/indirectly controlling — e.g. linear acceleration, roll or pitch. $\delta_{A_{sp}}$, $\delta_{E_{sp}}$, $\delta_{R_{sp}}$ corresponds to a negative factor of τ_x , τ_y and τ_z respectively. However, $\delta_{T_{sp}}$ does not correspond to a negative factor of T . Instead, those four terms determine the thrust required to apply inside the mixer in Figure 2.5. Having made this notice, it is of no importance to us as we use it as a black-box system.

The (translational) position and velocity controllers in Figure 2.6 are pretty straightforward. The former uses a P controller, whereas the latter employs a PID controller with a low-pass filter to reduce the noise received from the gyroscope. In PX4 documentation, it is noted that this low-pass filter on the derivative path is applied at the gyroscope driver side, so the controller software doesn't apply any additional filtering. Its explicit illustration is necessary for understanding the literate details of the controller scheme, though. Consequently, a translational acceleration setpoint $-A_{sp}$ is generated.

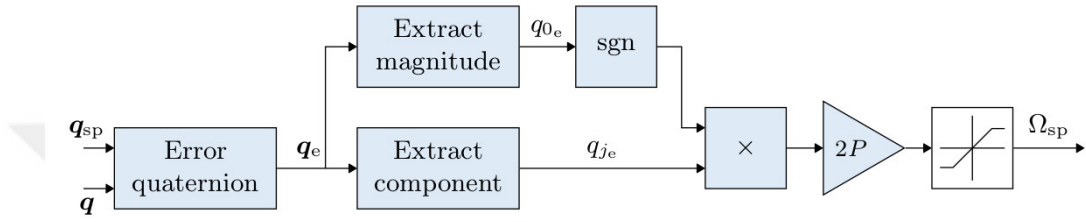


Figure 2.7: [Multicopter Attitude Controller], (n.d.), *Controller Diagrams*, https://docs.px4.io/master/en/flight_stack/controller_diagrams.html

Inner control loop -in the direction of increasing frequency, controllers are considered to be more *inside*- uses an attitude controller as its first component. This attitude controller is significantly different than the one in Fig. 4.21 of [14], which is the critical component that makes the control of aerial manipulator possible with PX4 framework. In a nutshell, the attitude controller in Fig. 4.21. of [14] uses Euler angles to compute the necessary roll and pitching torques to control the quadcopter at the hovering point. However, Euler angles are only locally stable in the non-linear global attitude control, i.e. the linearization applied hover point satisfies the dynamic constraints in a strictly limited angular position, velocity and acceleration manifold. Also Euler angles are subject to singularities, which have to be avoided. Nevertheless, it is not an actual problem in our case since the aim of this thesis is not to conduct extremely agile maneuvers typically seen in racing drones. To put it differently, the standard attitude controllers that use Euler angles to represent rotation do work solely with small roll and pitch angles. Therefore, they would basically result in the failure

of the drone for the aerial manipulator because the inherent asymmetry in the geometry causes much larger roll and pitch errors. That is why the global quaternion representation of rotations is used in the attitude controller of Figure 2.7, which is free from singularities. Since quaternions provide global asymptotic stability, a controller built on top of them could theoretically recover from *any* attitude to hovering state [20], which is the exact case when the disturbances due to the mere existence of manipulator cause larger errors in roll and pitch axis of the robot. This controller ultimately computes an angular acceleration setpoint to feed into the Angular Rate Controller.

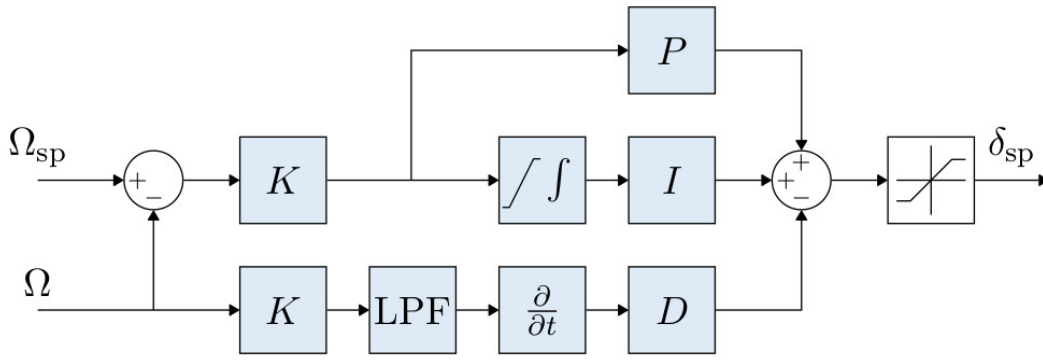


Figure 2.8: [Multicopter Angular Rate Controller], (n.d.), *Controller Diagrams*, https://docs.px4.io/master/en/flight_stack/controller_diagrams.html

This controller uses a PID scheme and calculates the elements $(\delta_{A_{sp}}, \delta_{E_{sp}}, \delta_{R_{sp}})$ necessary to compute rotor velocities. The mixer in Figure 2.5 handles the rest.

2.1.2 Control at Hover conditions in Gazebo

PX4 Control in Gazebo requires a set of sensor, actuator and communication interface plugins:

- *Multi-rotor base plugin*: The main component that fetches motor publication topic and maps them to existing rotors in initialization. In each iteration of Gazebo simulation, it publishes joint states of the rotors.

- *Motor model plugin*: This plugin simulates the physics of a rotor with some default parameters such as drag coefficient, motor constant, etc. It subscribes to a motor speed ROS topic, which is published by the *MAVLink Interface plugin*, to simulate the rotor motions in Gazebo.
- *GPS, Magnetometer and Barometer plugins*: Simulates respective sensors with parametrized noises.
- *MAVLink Interface plugin*: This is the critical component that creates the connection between PX4 and Gazebo. It takes all sensor topic names as parameters and submits all the messages published in them in appropriate MAVLink format to the emulated serial port (a port on localhost in our case, as in Fig. 2.3). Vice-versa, it takes the actuation commands, which are generated periodically by PX4 according to sensory inputs, from another emulated serial port and publishes them to motor speed topics.
- *IMU plugin*: Simulates inertial measurement unit. Most essential component for localization and estimation that exposes noise parameters for different constructs.

IMU plugin is deliberately mentioned as the last component since its parametric noise model fundamentally alters the takeoff behavior of the aerial manipulator. None of the raw sensor data is considered as ground truth by PX4 autopilot, but instead fused with the specified noise models with different weights in ECL EKF [21]. As the noise model becomes radical -i.e. the sensor claims its readings to be more accurate- Extended Kalman Filter would have a more challenging time correcting the inaccurate estimations at initialization. Contrarily, large noise models allow the filter to correct estimations earlier. However, this comes at a price. As the noise model of the sensor remains large, the estimations during the robot operation have a lower bound of confidence and always report the robot to be in some Gaussian volume -typically 10-15 cm in all axes, which is unacceptable for manipulation tasks that only allow a maximum of 1-2 cm of error in any axes. The default and zero noise models are respectively:

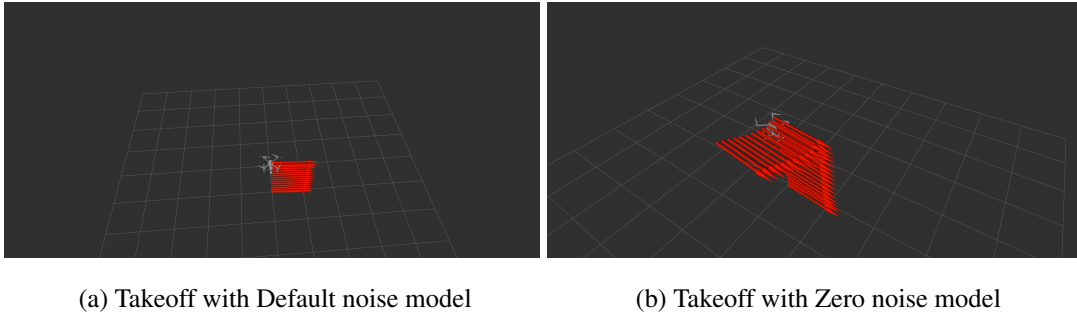


Figure 2.9: Takeoff behavior with different noise models in Rviz

```

<!-- Default noise model -->
<noise_model>
  <gyroscopeNoiseDensity>0.00018665</gyroscopeNoiseDensity>
  <gyroscopeRandomWalk>3.8785e-05</gyroscopeRandomWalk>
  <accelerometerNoiseDensity>0.00186</accelerometerNoiseDensity>
  <accelerometerRandomWalk>0.006</accelerometerRandomWalk>
</noise_model>

<!-- Zero noise model -->
<noise_model>
  <gyroscopeNoiseDensity>0.0</gyroscopeNoiseDensity>
  <gyroscopeRandomWalk>0.0</gyroscopeRandomWalk>
  <accelerometerNoiseDensity>0.0</accelerometerNoiseDensity>
  <accelerometerRandomWalk>0.0</accelerometerRandomWalk>
</noise_model>

```

2.1.3 Takeoff

Figure 2.9 shows the takeoff behaviors under different noise models of IMU. It is clearly visible that the default noise model results in a much better takeoff, whereas the zero noise model consistently results in a drift along x-y axes before recovering back to the target altitude. Before settling on default noise model based on this comparison alone, it is obligatory to conduct more analysis at the hovering conditions on

idle, full rotation around z_B axis and under manipulation effect cases.

2.1.4 Full Rotation

In this test case, the aerial manipulator is commanded a rotation of 2π around z_B in clockwise direction with different noise models.

The default noise model in Figure 2.10a causes the robot to fluctuate significantly along x , y and z axes, which leads to increased settling time or worse, the loss of translational component of the 3D pose — i.e. failing to settle on the previous translational component. The range of drift along z axis is 15 centimeters for instance, which is unacceptable for the fruit-picking scenario. On the contrary, the zero noise model in Figure 2.10b provides a bounded fluctuation of 4 centimeters. Therefore, the zero noise model is much better for the target scenario and even within the cartesian error limits for the desired manipulation task.

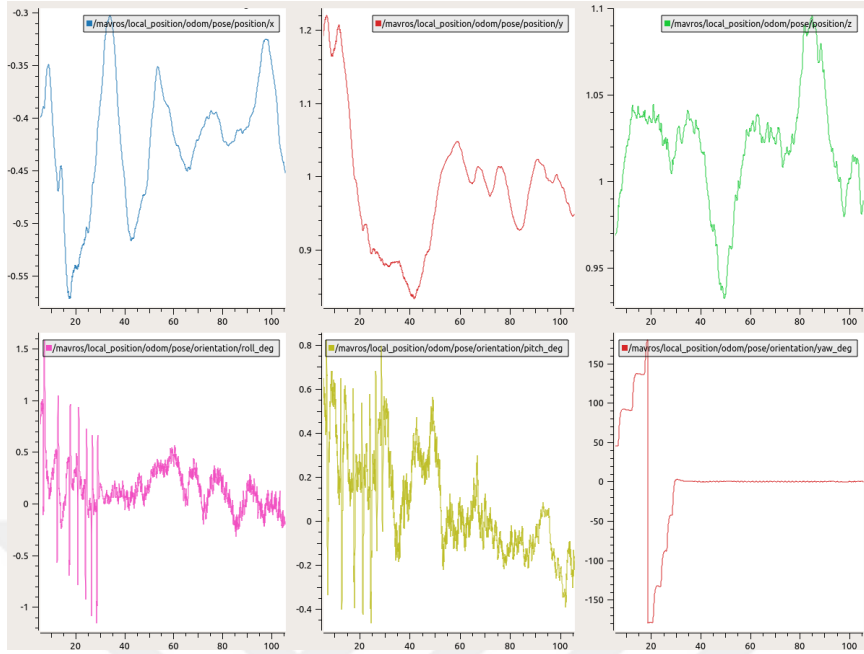
2.1.5 Idle

Estimated states of the aerial manipulator are observed at hover conditions with different noise models in this test case.

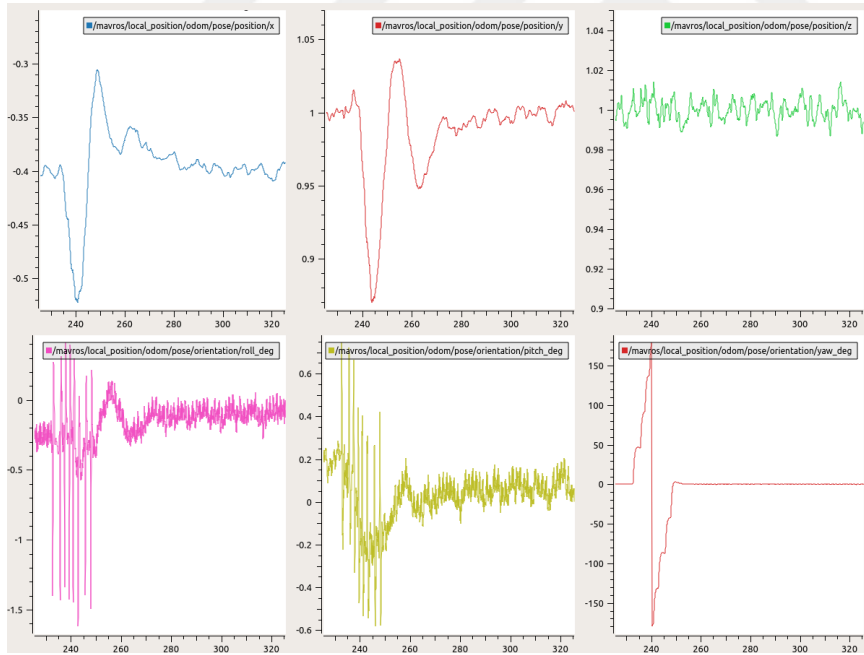
The uppermost plots in Figures 2.11 and 2.12 are almost identical to the previous test case. On these alone, we can conclude that the zero noise model is much better with respect to the default noise model despite the drifting takeoff behavior. The noise models with relatively smaller amplitudes can be directly observed in these plots on the linear acceleration along x_B and z_B axes. The linear acceleration along y_B is due to the attached manipulator, which is positioned along y_B axis.

2.1.6 Under Manipulation Effect

The final test case is the most important one as it studies the effects of the manipulator to aerial base at hovering conditions.



(a) Full Rotation with Default noise model



(b) Full Rotation with Zero noise model

Figure 2.10: Robot responses to 2π rotation around z_B with different noise models

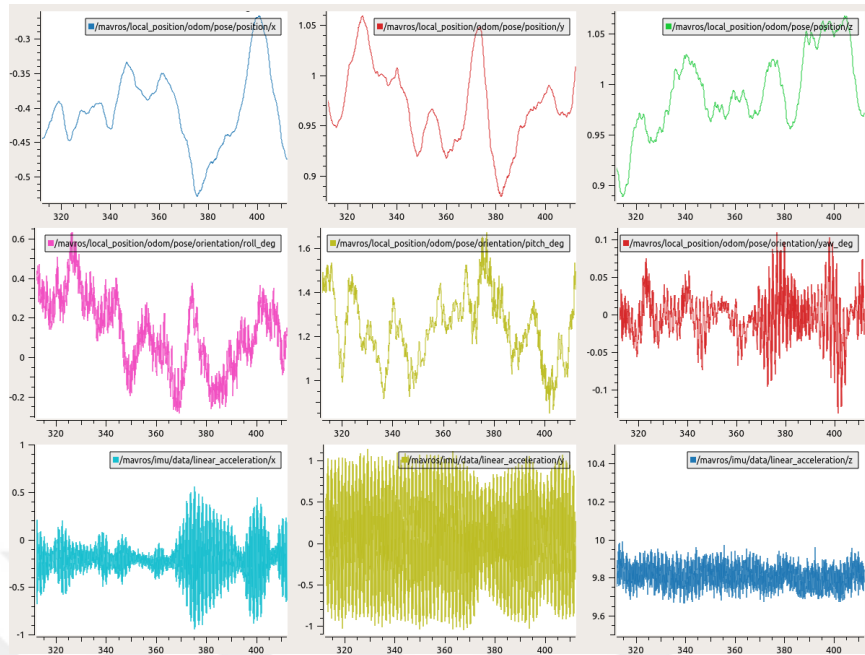


Figure 2.11: Idle with Default noise model

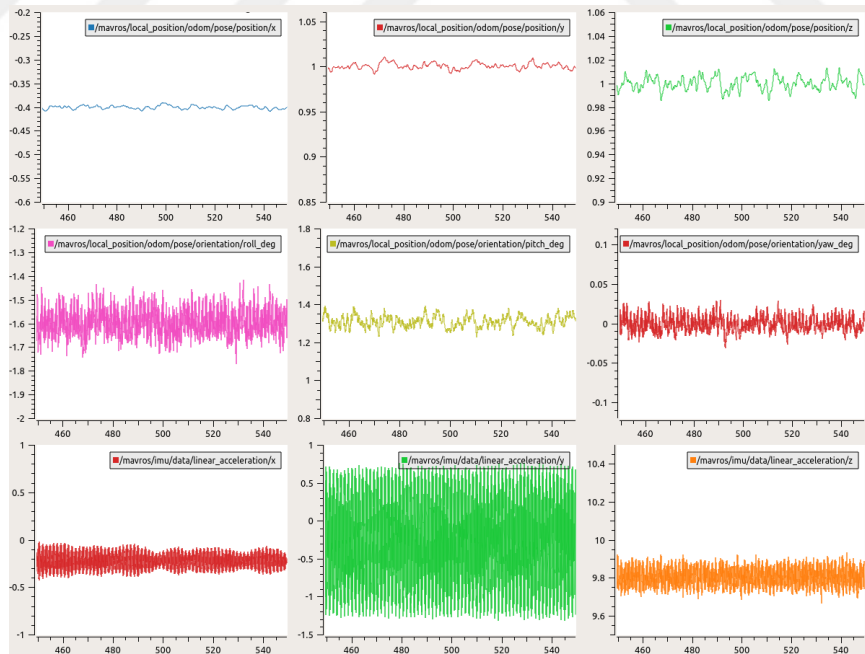
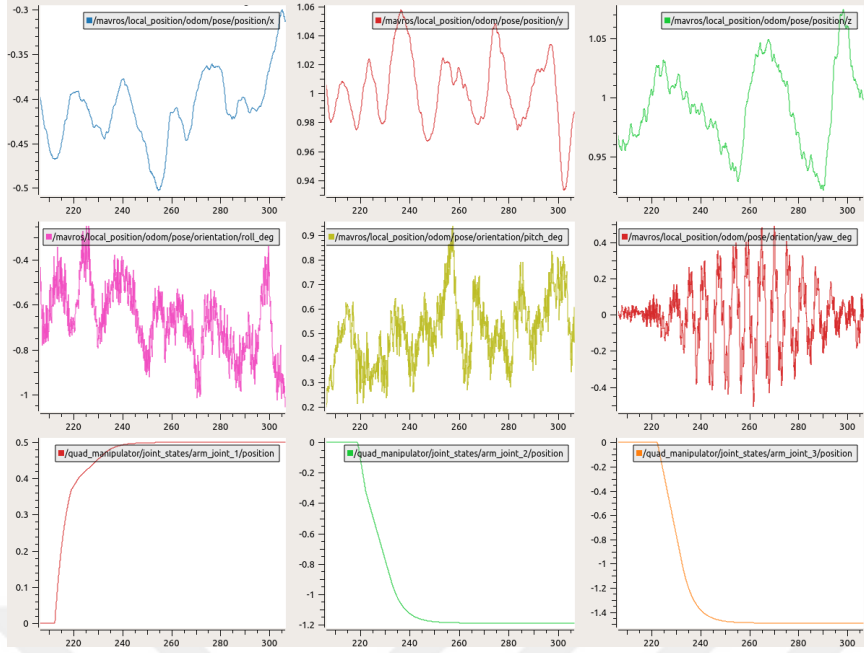
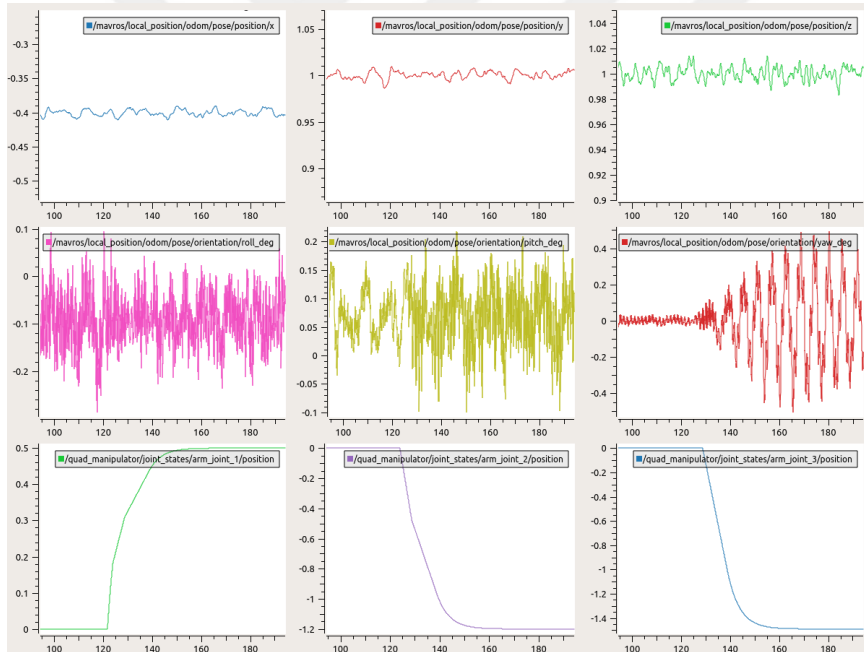


Figure 2.12: Idle with Zero noise model



(a) Under Manipulator Effect with Default noise model



(b) Under Manipulator Effect with Zero noise model

Figure 2.13: Robot responses to manipulator motions with different noise models

The attached manipulator executes a trajectory from $q^s = [0.0, 0.0, 0.0]$ to $q^f = [0.5, -1.197, -1.495]$ in 20 seconds. Again translation of the robot is kept as is in the

zero noise model, whereas Fig 2.13a sketches once again an unreliable hover profile for the fruit picking scenario. The fluctuations in yaw degree are negligible in both plots. Motions of the aerial manipulator are simulated in Gazebo and recorded for default and zero noise processes respectively at <https://drive.google.com/file/d/1M1tb76-6LeKKmNNncWYSfOfaZ-BFP1wY> and <https://drive.google.com/file/d/1lSNrNQges5YXQ6f0uD8O7FB7NudpVIAx>.

Based on these illustrative facts, the zero noise process model is the reasonable option to choose, wherein the drifting takeoff is the inevitable tradeoff. However, as the robot will conduct its takeoff operation in a wide-open base location, it is not a problem for the application targeted in this thesis.



CHAPTER 3

PLANNING

We use a particular family of motion planning algorithms, *namely Rapidly Exploring Random Tree(RRT)*[22], in a multi-robot setting where each robot is commanded to a final location. The goal of these algorithms is to produce feasible motion profiles for the respective final configurations. Ultimately the most successful algorithm is selected according to the evaluation criteria from the overall benchmark.

Typical PEAS model requires the explicit definition of the following:

- Performance Measure
- Environment
- Actuators
- Sensors

In the beginning, 2D and 3D fully kinematic worlds are used to benchmark the algorithms; hence the model can be defined respectively:

- Three criteria: *Path cost, Completion percentage, Execution time*
- 2D, 3D Cartesian worlds with spherical obstacles.
- No actuator. RRT planners are agent-agnostic.
- Communication pseudo-sensor: the environment is completely known *a priori*.
The robots communicate through intra-process communication.

The model definition surfaces some of the intrinsic features of the RRT planners. Such that, they are originally *offline*, *static*, *global*, *sub-optimal* and *incomplete* algorithms. Besides, they assume a fully observable environment (that is, the *a priori* map is correct and exact). All these undesirable features are compensated with the efficiency and speed introduced through the randomization (i.e. sampling-based nature) of RRTs. Almost every study in the literature addresses some subset of the above characteristics for their improvement (e.g. having an online, dynamic RRT[23] algorithm, claiming asymptotical optimality[24] or introducing a deterministic sampler for *completeness*[25]) to better adapt the algorithm to real-world scenarios.

Four different algorithms have been implemented for the single-robot case: RRT[22], RRT*[24], Bidirectional RRT*[26] and Bidirectional RRT* with a heuristic. It is cumbersome to rewrite all these algorithms' pseudocodes here again; therefore, it is sufficient to explain how they work all in all. Since they extend each other in a building block manner, discussion on the extension functions should enlighten their respective mechanisms.

Multi-robot path planning requires a much more sophisticated algorithm than the above ones to resolve robot-robot collisions. There are three different coordination methodologies regarding the resolution of multi-robot path planning problem, each having different trade-offs:

- i **Coupled/Global:** This class of methods searches over the composite state space of all the agents, treating them as a unified meta-agent. It is clear that the state-space complexity is exponential in the number of agents; thus, the global methods can only be applied in the cases where the number of robots is relatively low. Nevertheless, there are methods that embrace global techniques to attack multi-robot path planning problem by introducing special data structures with efficient search metrics.[27]
- ii **Decoupled/Local:** In these methods, paths are planned by the agents separately. A coordination/cooperation strategy repairs when the robot-robot collisions occur among them. Decoupled approaches cannot guarantee completeness unless sup-

ported by global factors. Due to their decoupled design, the computation complexity does not differ much from the single-robot planners. Two examples of repairing methods are *Velocity tuning*[28] and *Sub-dimensional Expansion*[29].

- iii **Prioritized:** These methods also do plan paths separately, but integrate the paths of agents with higher priorities as moving obstacles into their configuration space. They cannot guarantee completeness either, since the previously planned paths of higher priority agents might make the solution impossible for the lower priority agents. Re-prioritization schemes are developed to address this inherent problem, but they are out of scope here, since we will not embrace a priority-based coordination technique.

Sub-dimensional Expansion[30] and sRRT[31] are the technique and the algorithm that will be implemented in the multi-robot part of this study; thus, therein will be explained in more depth.

3.1 Single UAV Planning

3.1.1 RRT

This algorithm provides the basis of the other three variants. Before delving into their mechanisms, it is imperative to eliminate possibly a slight confusion. RRT is, in fact the underlying data structure, *i.e. fringe*, and the algorithm is essentially an uninformed constrained search method that is populating this fringe in a non-deterministic way, given that a random sampling method is used. Nonetheless, *RRT-search* term is not used thus far in the literature as we know of and the convention is on using RRT, which convolves the algorithm and the data structure.

RRT's overall mission is finding a collision-free path from a starting configuration x_{start} to the goal configuration x_{goal} . The groovy task of RRT in each iteration is to simply sample a random configuration x_{rand} in the given bounded search space. However, unintuitively, instead of trying to connect this random configuration directly

to the tree (making the nearest node(configuration) x_{near} in the tree its parent), it does the following steps:

1. finds x_{near} , computes the control input u that would drive the conceptual agent¹ from x_{near} to x_{rand} .
2. Applies u to the x_{near} for a Δt and observes resultant configuration x_{new} .
3. Asks the underlying collision structure that whether x_{new} collides with any obstacle. If not, it proceeds to check the intermediate configurations² between x_{near} and x_{new} . If the path is collision-free, then x_{new} is added to the tree.

The above steps are repeated until x_{goal} is added to the tree or the finite number of iterations is exceeded. From this definition, it is seen that the algorithm is incomplete. In the following subsection, its *almost-sure sub-optimality* is investigated.

3.1.2 RRT*

RRT* is the asymptotically optimal variant over RRT suggested by [24]. *Theorem 21* of the respective study gives legal proof on the asymptotical sub-optimality of RRT and the asymptotical optimality of RRT*. Their contribution through RRT* is the famous *rewire* method, which operates on the inserted nodes of the tree and rewires a finite number of them through changing the parents of the inserted nodes to the newly added possibly lower-cost node. Through this method, the algorithm ensures asymptotical optimality (as $i \rightarrow \infty$ the computed path is *optimal*). Asymptotical optimality has another advantage in terms of *completeness*. Since the algorithm provides *almost-sure optimality*, it means that it can find a solution if there is any. Conjunctively this feature translates to *probabilistic completeness*. Overall, RRT* is a *superior* algorithm to RRT since it only adds a constant overhead and proposes optimality and completeness quantitatively.

¹ Remind that it is originally an offline algorithm, so the agent is not moving in fact.

² since the space is continuous, discretization is imperative as a simple, iterative method

3.1.3 Bidirectional RRT*

The algorithms described thus far employ only one instance of the underlying tree data structure. Nonetheless, it is not a strict requirement and even employing multiple data structures and proposing an intelligent connection mechanism improves significantly over RRT* as BiRRT* does. In detail, it initializes two trees, one from the x_{start} and the other from the x_{goal} . In each iteration, only one tree is selected for sampling either deterministically or non-deterministically. After inserting x_{new} in the selected tree and applying `rewire` method, the nearest neighbors from the other tree are queried. If there exists any neighbor node with a collision-free path to x_{new} , x_{new} is added to the other tree, being the neighbor node as its parent and eventually, the trees become connected. If there are multiple neighbors with collision-free paths, the one with the least-cost path is selected as the parent. Otherwise, the algorithm continues to the next iteration. Typically the algorithm does not finish after finding a valid path σ_{best} and maintains the loop with the hope of finding a better path. This feature can be thought of as the critical factor of asymptotical optimality.

3.1.4 Bidirectional RRT* with Heuristic

The heuristic introduced in this algorithm is a relatively simple one and would significantly reduce the completion time. The heuristic work on σ_{best} by trying to shorten it by eliminating intermediate nodes between two randomly selected and nonconsecutive nodes. In fact, it works as an unbounded `rewire` function, in which all the nodes in the path are considered neighbors. It is definitely an *admissible* heuristic, but the improvement introduced by it might not even compensate for the computational overhead. At first glance, it can be thought of as a good heuristic, but the hyper-parameter Δt already constrains the distance between nodes and the `rewire` operation with a proper *rewire count* (another hyper-parameter) handles the sub-optimal intermediate node connections all along.

3.1.5 Implementation

The implementation is done purely in C++ and the algorithms are bundled as a library for opening to the public later on. In order to sustain computational efficiency, R-tree spatial data structure, which is a self-balanced tree that can employ different node splitting algorithms, e.g. linear and quadratic, is used in preliminary work. But a thorough research [32] reveals that octrees³ are better candidates for spatial shape registry, especially in higher dimensions and will be used instead of R-tree in the subsequent works. Also, the current implementation exports the responsibility of collision knowledge, i.e. entities in the world other than sampled configurations, to an interface that can be implemented in any way desired, e.g. through geometric shape registry or keeping 2D-3D occupancy grids/maps. Would the OctoMap method is embraced, it should be embedded into this agent-agnostic (also, the agent is agnostic about it, too) interface as an internal structure so that any type of collision information introduced by the user is translated into it.

After implementing the algorithms, one minor improvement is added. Such that, all algorithms employ an early greedy path calculation (discretized with Δt from x_{start} up until x_{goal}) in the entrance of their `plan` methods. If the greedy path is collision-free, it is returned by the algorithms as the solution. Nicely, the *greedy choice* results in optimal solutions (the shortest path between two points in Euclidean/geodesic spaces) similar to A* and Dijkstra.

Ultimately RRT library is implemented using Template Meta-Programming, so theoretically, it can be made any-dimension (Nd). The underlying data structure determines the practical limitation for the algorithms to remain functional.

3.1.6 Evaluation - Metrics

Evaluation criteria are based on three metrics as described above:

³ MoveIt motion planning framework uses OctoMap.(could be considered as a 3D octree)

- Path cost
- Execution time.
- Completion Percentage

In order to analyze the true nature of the algorithms, the greedy-choice property is disabled during the performance tests. The following figures represent sample runs in 2D cartesian worlds $W \in [0, 100] \times [0, 100]$.

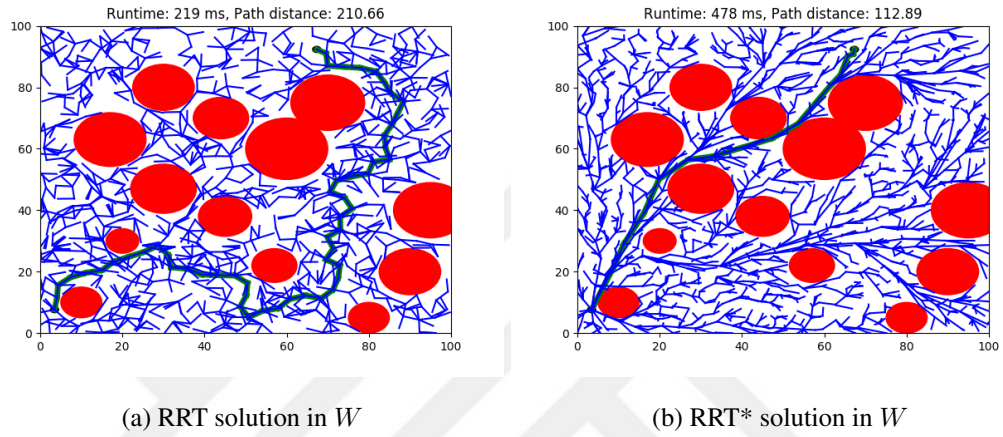


Figure 3.1: RRT Algorithms

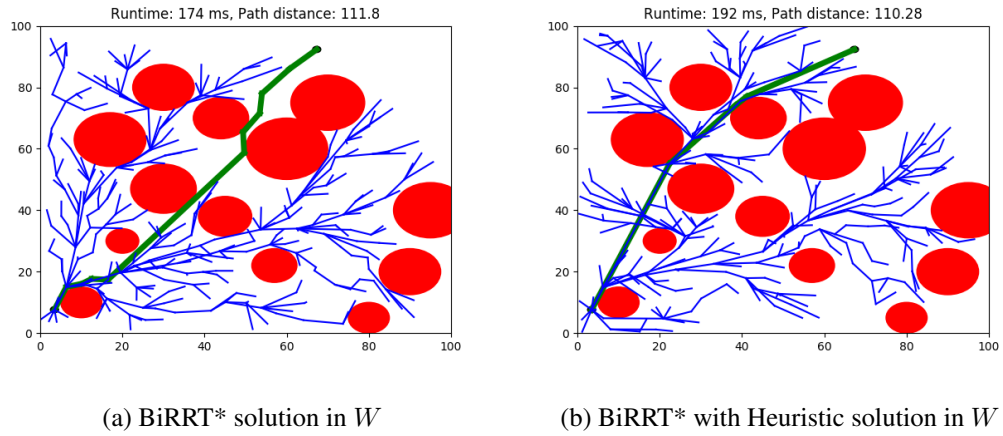


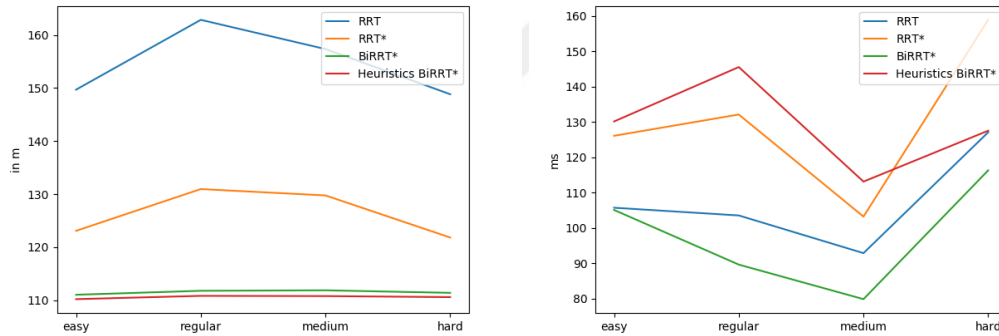
Figure 3.2: RRT Algorithms - Continued

In order to objectively test the algorithms' success rate, four different environments

are designed: *Easy*, *Regular*, *Medium* and *Hard* having respectively 6, 9, 12 and 15 circle obstacles. In each environment, all the algorithms ran 500 times and the run-time and path-costs are averaged. The common hyper-parameters for all four algorithms are the same and can be found in Table 3.1.

Table 3.1: Common RRT Hyperparameters

Common RRT Hyperparameters	
x_{start}	(3.47, 7.63)
x_{goal}	(67.31, 92.34)
Max. samples	2000
resolution	1.0
delta_q	4.0
rewire_count	3



(a) Average path cost of four algorithms in 200 runs (b) Average execution time of four algorithms in 200 runs

Figure 3.3: RRT Metrics

Figure 3.3 reports two metrics: average path cost and average execution time. It can be seen from 3.3a, the heuristic method did help very little if any and the slight improvement on path-cost seemingly traded off with the execution time as shown in 3.3b. Furthermore, introducing a second internal data structure, as BiRRT* does, improves the path-cost more than expected. This phenomenon can be attributed to the

Table 3.2: RRTs Success Rate

	Success Rate			
	Easy	Regular	Medium	Hard
RRT	1.0	1.0	1.0	0.794
RRT*	1.0	1.0	1.0	0.774
BiRRT*	0.996	0.978	0.996	0.966

biased extension of both trees to each other, which is a genuinely clever method for approximating optimal solutions through promoting direct line distances at least on some portion of the path. Finally, the heuristic used employs a very similar method as described above and hopes to address some room for optimization along the computed path, eventually improves just a little on it. On the contrary, the execution time exceeds its uninformed counterpart's execution time, i.e. BiRRT*, in **hard** environment.

Since all these algorithms are incomplete by their nature and some of them are *probabilistically complete* at most, their success rates need to be computed. However, even in the hard environment, they all had a 1.0 success score, i.e. they have found a valid solution in all the runs. In order to produce meaningful success scores, the maximum number of samples is reduced to 300. The resultant success scores are as follow:

The heuristic version is left out from this metric since it is just the same with BiRRT* in terms of completeness. From Table 3.2, it can be seen that BiRRT* has a significant superiority over RRT and RRT*.

3.2 Multi-UAV Planning

3.2.1 Sub-dimensional Expansion

Sub-dimensional Expansion is a multi-robot path planning framework that performs well, given that the interactions between robots are sparse. This condition allows a natural decoupling in the planning problem:

1. Plan individually,
2. Resolve conflicts in the individual plans optimally.

Since it is a framework, it could be applied to any type of state-space definition, either implicit (e.g. M^* [30]) or explicit (e.g. sRRT[31]). As enumerated above, the general framework starts by computing *individual policies* per robot. Those individual policies only ensure robot-object collision avoidance. Each robot contributes its own policy to the multi-robot system, which results in the *joint policy path*, where the robot-robot collisions are likely to present. The technique behaves *optimistically*, i.e. assumes that the joint policy path is collision-free until explicit evidence exists; hence, each robot strictly follows its own policy initially. An underlying planner processes the joint policy path until finding a robot-robot collision. When such a condition is encountered, the involved robots are permitted to diverge from their individual policies through increasing the dimensionality of search space locally. In that local search space, joint actions of the respective robots are planned in a coupled way to resolve robot-robot collision(s). The true strength of this technique comes from its hybrid nature, since it employs the tractable decoupled planning initially and scales up the involved dimensionality only when necessary and still with lower complexity than full configuration space as in the global methods, where all agents are treated as one meta-agent.[30]

3.2.2 sBiRRT*

The previous section discussed the Sub-dimensional Expansion framework, which any planning algorithm can employ with certain caveats. sBiRRT* is a novel al-

gorithm that builds on top of sRRT[31], which uses Sub-dimensional Expansion to guide the construction of a forward-growing tree for multi-robot systems in joint configuration-state. sBiRRT* internally uses BiRRT* planners for each robot to construct asymptotically optimal individual policies; therefore, agent-specific robot-object collision-free BiRRT* solutions are termed as *policy trees*[29]. It is suggested in the original paper to grow the tree backward; nonetheless, it is currently a no-op in our work since the implementation already supports an auxiliary fringe for reaching parent from child, which exploits the unique parentage feature of nodes in trees by construction. This relationship is formally defined as in the following:

$$\phi(q_k^i) = p_k^i \quad (3.1)$$

Following identities are also valuable for understanding the general layout of the algorithm:

- $q_s = q_s^1 \times \dots \times q_s^n$, where q_s^i corresponds to the start state for robot r^i .
- $q_f = q_f^1 \times \dots \times q_f^n$, where q_f^i corresponds to the goal state for robot r^i .
- The left-hand-side expressions in the above equations are in fact *joint state configurations*. Therefore q_k^i corresponds to the configuration of r^i specified by q_k .

Algorithm 1 Collision Set Backpropagation

```

1: procedure BACKPROP( $q_k, C_k$ )
2:   if  $\phi(q_k)$  is  $\emptyset$  then
3:     return ▷ Base case
4:   end if
5:    $p_k, \_, C_j \leftarrow \phi(q_k)$ 
6:   if  $C_j \not\subseteq C_k$  then ▷ If collision-set of parent is already a superset of its child
     then no need to back-propagate more.
7:      $C_j \leftarrow C_j \cup C_k$ 
8:     return BACKPROP( $p_k, C_j$ )
9:   end if
10: end procedure
11: 
```

The **for-loop** starting at line 28 in Algorithm 2 is the place where Sub-dimensional Expansion is implicitly occurring. If a robot r^i is an element of C_k , then it means it is

Algorithm 2 sBiRRT*

```
12: Let  $R = \{r^1, \dots, r^n\}$  be distinct robots.
13: Let  $q_s^i$  and  $q_f^i$  be the start and goal configurations of the robot  $r^i$  respectively.
14: Let  $P_{single}$  be BiRRT* planner in single configuration-space.
15: Let  $P_{expanded}$  be BiRRT* planner in joint configuration-space.
16: Let  $\sigma_{kl}$  represent the path starting from  $q_k$  to  $q_l$  (inclusive).
17: // Compute individual policy trees.
18: for all  $r^i$  do
19:    $T^i \leftarrow \{(q_s^i, \phi(q_s^i), C_s), \dots, (q_k^i, \phi(q_k^i) = q_f^i, C_k), (q_f^i, \emptyset, C_f)\}$   $\triangleright$  Note that  $C_k$ 
   (collision sets) are defined in the joint configuration-space.
20: end for
21: Let  $T_f$  be the forward-growing tree in the joint configuration-space.
22:  $T_f \leftarrow \{q_s\}$ 
23:  $l \leftarrow 0$ 
24: while  $q_f \notin T_f$  or  $l > l_{max}$  do
25:   Randomly sample  $q_z \in Q$ , where  $Q$  is joint configuration-space.
26:    $(q_k, p_k, C_k) \leftarrow nearest(T_f, q_z)$ , where  $p_k$  and  $C_k$  are directly coupled with
    $q_k$ .
27:   Let  $q_z^i$  be the sample in the search space  $Q^{sch}$ 
28:   for all  $r^i$  do
29:     if  $r^i \in C_k$  then
30:        $q_z^i \leftarrow q_z^i$ 
31:     else
32:        $q_z^i \leftarrow p_k^i$ 
33:     end if
34:   end for
35:   // Pass-by planning. Can be done in two passes.
36:    $\sigma_{kf} \leftarrow P_{expanded}.plan(q_k, q_z^i, q_f)$ 
37:   if  $\sigma_{kf} = \emptyset$  then
38:     continue
39:   end if
40:    $\sigma_{sf} = \sigma_{sk} \cup \sigma_{kf}$   $\triangleright$  Retrieve  $\sigma_{sk}$  from policy trees.
41:   for all  $r^i \mid \exists j (\sigma_{sf}^i \cap \sigma_{sf}^j \neq \emptyset) \ \& \ r^i \notin C_k$  do  $\triangleright$  where  $r^j \in R$ 
42:      $C_k \leftarrow C_k \cup r^i$ 
43:   end for
44:   BACKPROP( $q_k, C_k$ )
45:   for all  $q_j$  in  $\sigma_{kf}$  do
46:     if  $(q_j, \phi(q_j)) \cap \sigma_{sf} = \emptyset$  then
47:        $T_f \leftarrow T_f \cup q_j$ 
48:     end if
49:   end for
50:    $l \leftarrow l + 1$ 
51: end while
```

in a robot-robot collision case and needs to deviate from its individual policy with a new sample. Therefore the relative portion of joint configuration-sample q_z , i.e. q_z^i , is assigned to the relative portion of the projected sample. Otherwise the robot is not in a robot-robot collision case and can gracefully continue its own policy; therefore, the projected sample takes the parent of q_k^i , i.e. p_k^i . This can be expressed in an alternate way, which expresses the Sub-dimensional Expansion more intuitively. The robots that can follow their individual policies without robot-robot collisions are restricted to follow them, so they are not effectively sampled. Only those in collisions effectively sample and are allowed to go to a valid configuration without being restricted to their individual policies. After determining local search space sample q'_z , $P_{expanded}$ tries to plan a robot-object collision-free path in the joint configuration-space from q_k to q_f , while passing from q'_z . If the path σ_{sf} contains any robot-robot collisions, then the involved robots are added to C_k and this expanded collision-set is backpropagated to the parent node of q_k . This action signals the necessity of a higher-dimensional effective sampling in next iterations, when such nodes are determined to be the closest neighbors with respect to random sample q_z in joint configuration-space Q . Finally, the **for-loop** that starts at line 45 adds the edges of σ_{kf} to T_f if there aren't any robot-robot collisions⁴ with respect to σ_{sf} .

Figures 3.4-3.10 illustrate the machinery of the algorithm step-by-step, wherein the initial paths are produced with BiRRT* algorithms by each robot. Robot-robot collisions are also solved with a BiRRT* planner that runs on the joint configuration-state. The odd numbered steps are the phases where the collision set expansion occur – i.e. between lines 41 and 44 in Algorithm 2. The even numbered steps illustrate the expansion of T_f , which is done in the **for-loop** starting at line 45.

⁴ Note that σ_{kf} is already robot-object collision free if it is not empty.



Initialization

$$q_s = (q_s^1, q_s^2, q_s^3)$$

$$T_f = \{q_s\}$$

$$\phi^3(q_s^3) = q_a^3$$

$$\phi^1(q_s^1) = q_a^1 \quad \phi^2(q_s^2) = q_a^2 \quad \phi^3(q_a^3) = q_b^3$$

$$\phi^1(q_a^1) = q_f^1 \quad \phi^2(q_a^2) = q_f^2 \quad \phi^3(q_b^3) = q_f^3$$

$$\phi^1(q_f^1) = \emptyset \quad \phi^2(q_f^2) = \emptyset \quad \phi^3(q_f^3) = \emptyset$$

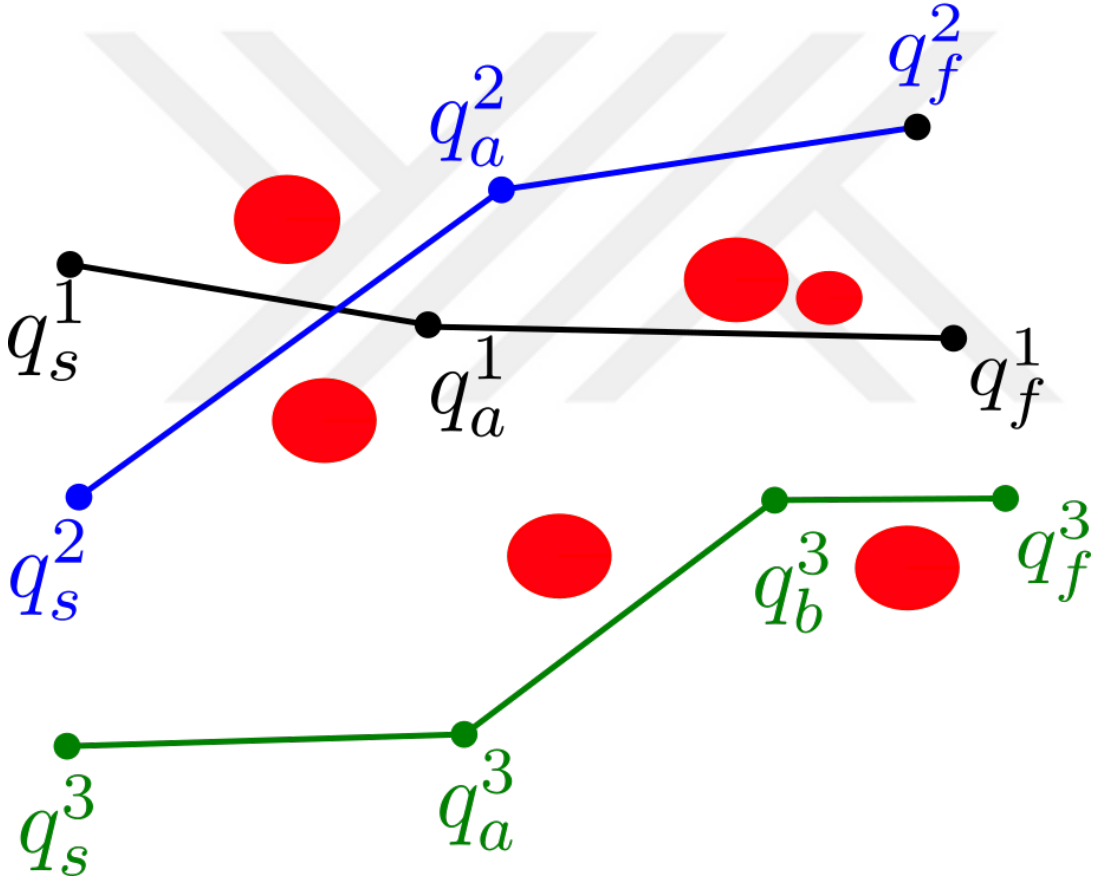
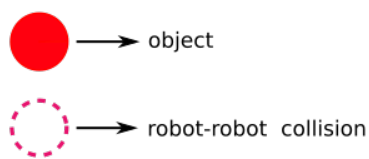


Figure 3.4: Initial state after retrieving policy tree for each robot

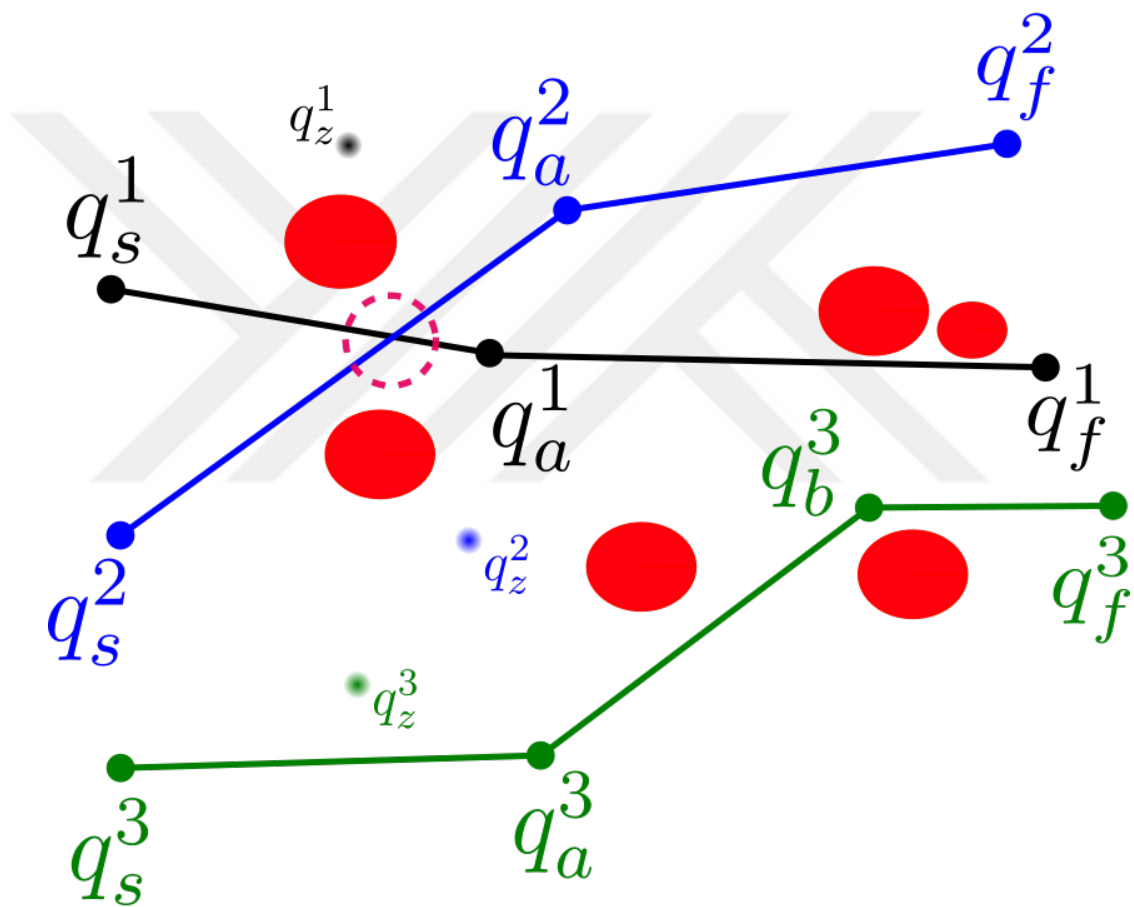
The algorithm starts working on the respective policy trees of each robot in Figure 3.5. In there, robot-robot collision between the policies of r^1 and r^2 are observed and thus added to the collision set of the only node in $T_f - q_s$. In each iteration, a vertex



Step-1

Pre-operation

$$q_k = (q_s^1, q_s^2, q_s^3) \quad C_k = \emptyset$$



Post-operation

$$C_k = \{r^1, r^2\}$$

Figure 3.5: sBiRRT* robot-robot collision in Step-1



Step-2

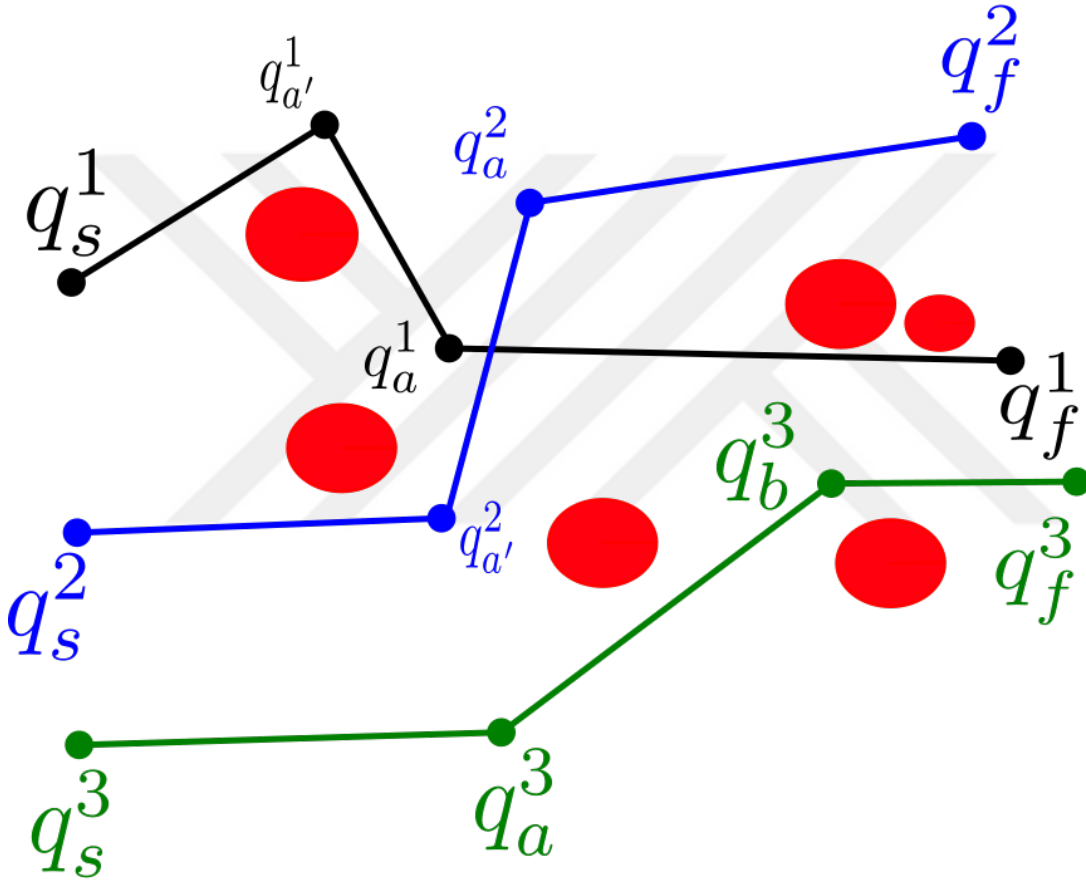
Pre-operation

$$q_k = (q_s^1, q_s^2, q_s^3)$$

$$T_f = \{q_s\}$$

$$C_k = \{r^1, r^2\}$$

$$q'_z = (q_{a'}^1, q_{a'}^2, q_a^3)$$



Post-operation

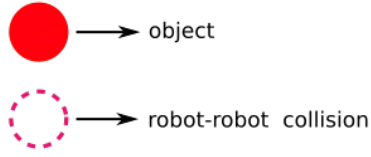
$$T_f = \{q_s, (q_{a'}^1, q_{a'}^2, q_a^3)\}$$

$$\phi^1(q_s^1) = q_{a'}^1, \phi^2(q_s^2) = q_{a'}^2,$$

$$\phi^1(q_{a'}^1) = q_a^1, \phi^2(q_{a'}^2) = q_a^2$$

Figure 3.6: sBiRRT* robot-robot collision resolution in Step-2

Step-3



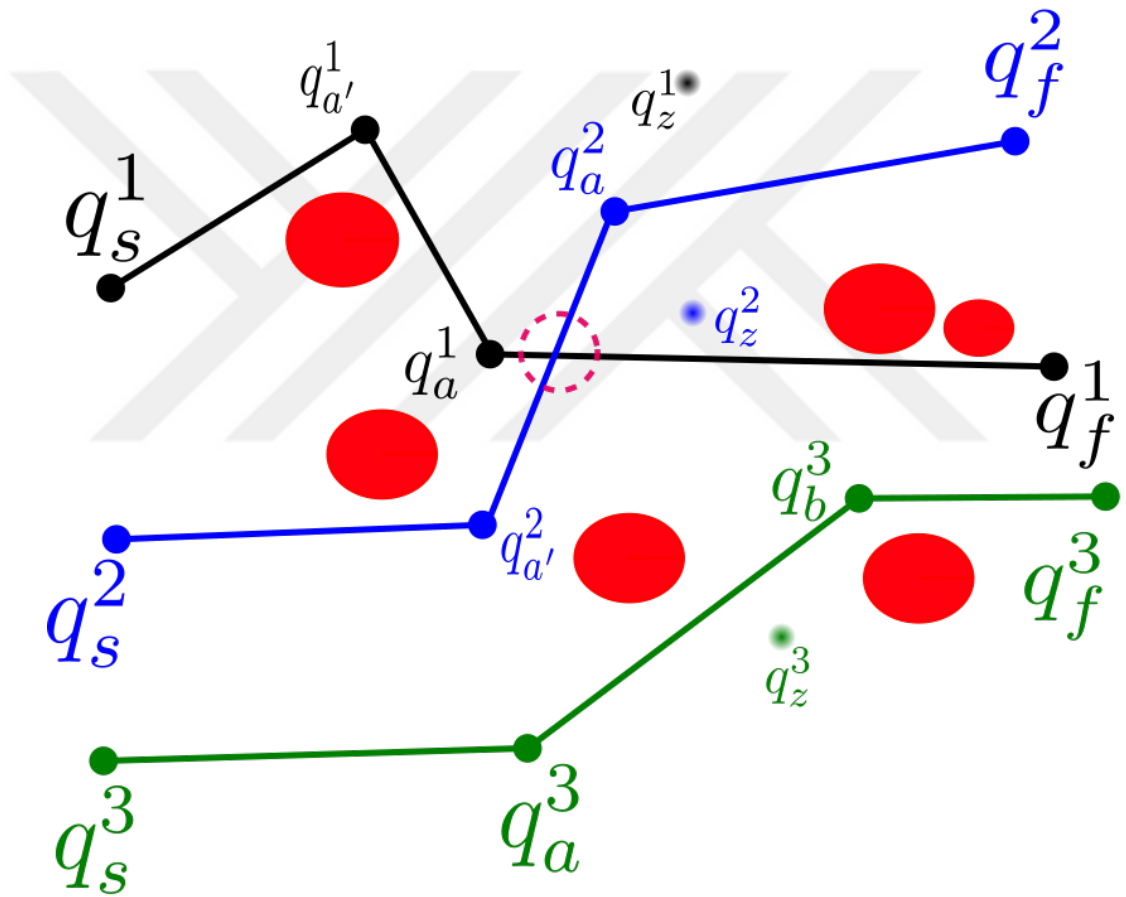
Pre-operation

$$q_k = (q_{a'}^1, q_{a'}^2, q_a^3)$$

$$T_f = \{q_s, (q_{a'}^1, q_{a'}^2, q_a^3)\}$$

$$C_k = \emptyset$$

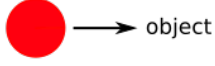
$$q_z = (q_z^1, q_z^2, q_z^3)$$



Post-operation

$$C_k = \{r^1, r^2\}$$

Figure 3.7: sBiRRT* robot-robot collision in Step-3



Step-4

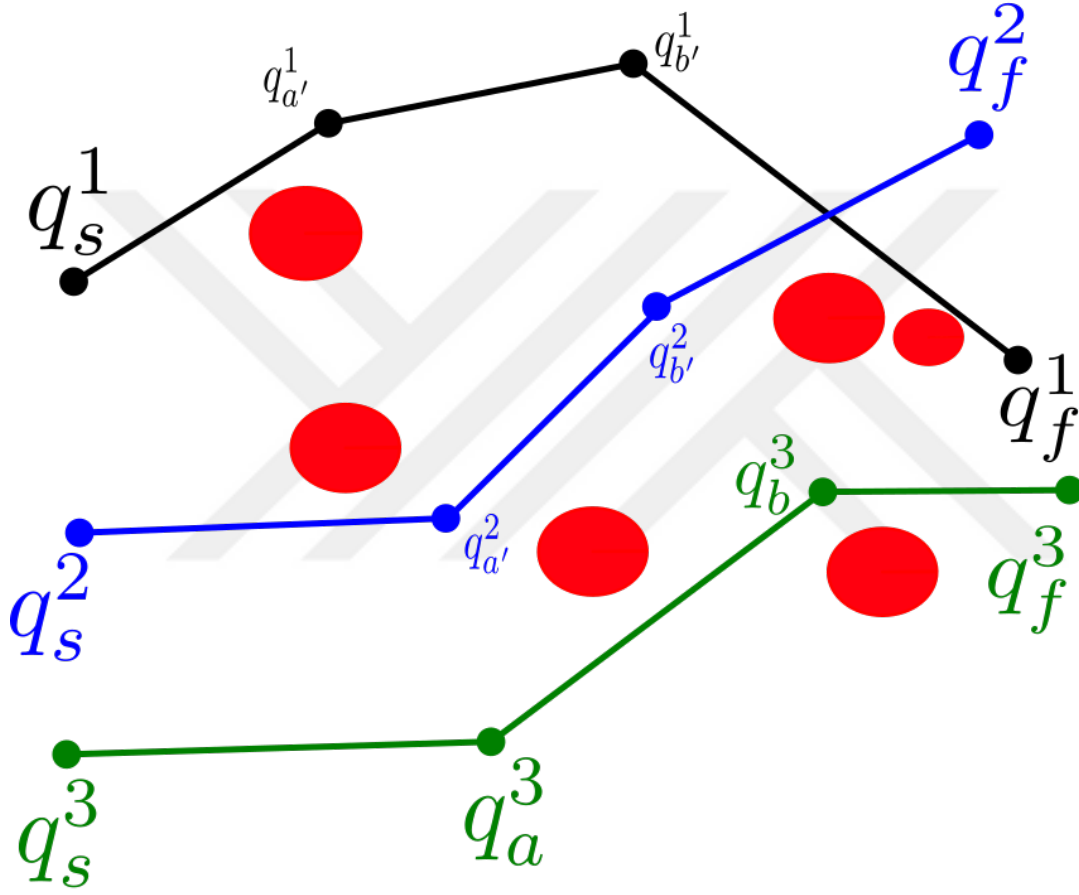
Pre-operation

$$q_k = (q_{a'}^1, q_{a'}^2, q_a^3)$$

$$T_f = \{q_s, (q_{a'}^1, q_{a'}^2, q_a^3)\}$$

$$C_k = \{r^1, r^2\}$$

$$q'_z = (q_{b'}^1, q_{b'}^2, q_b^3)$$



Post-operation

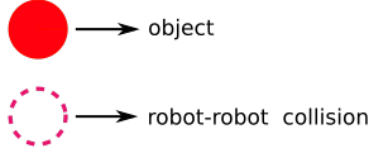
$$T_f = \{q_s, (q_{a'}^1, q_{a'}^2, q_a^3), (q_{b'}^1, q_{b'}^2, q_b^3)\}$$

$$\phi^1(q_{a'}^1) = q_{b'}^1, \phi^2(q_{a'}^2) = q_{b'}^2$$

$$\phi^1(q_{b'}^1) = q_f^1, \phi^2(q_{b'}^2) = q_f^2$$

Figure 3.8: sBiRRT* robot-robot collision resolution in Step-4

Step-5

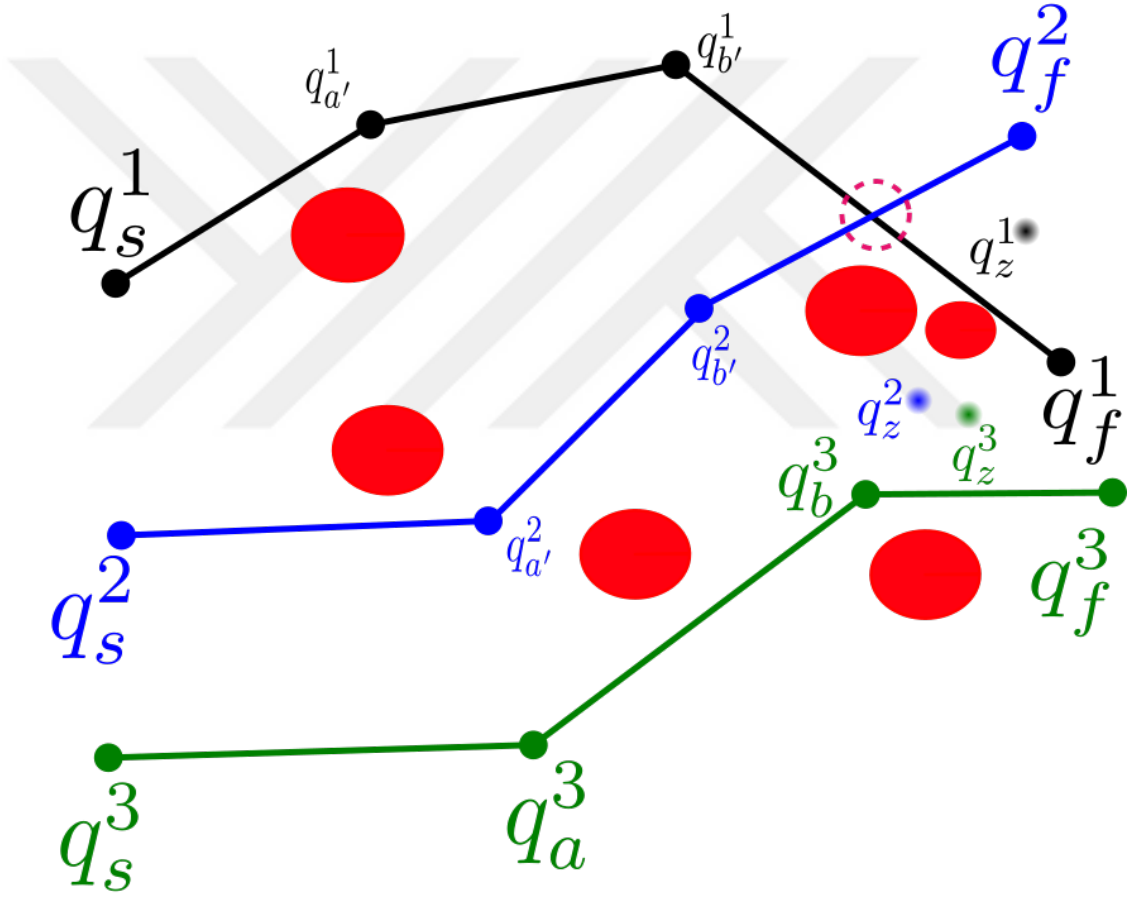


Pre-operation

$$q_k = (q_{b'}^1, q_{b'}^2, q_b^3)$$

$$C_k = \emptyset$$

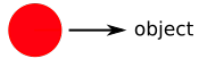
$$T_f = \{q_s, (q_{a'}^1, q_{a'}^2, q_a^3), (q_{b'}^1, q_{b'}^2, q_b^3)\} \quad q_z = (q_z^1, q_z^2, q_z^3)$$



Post-operation

$$C_k = \{r^1, r^2\}$$

Figure 3.9: sBiRRT* robot-robot collision in Step-5



Final Step

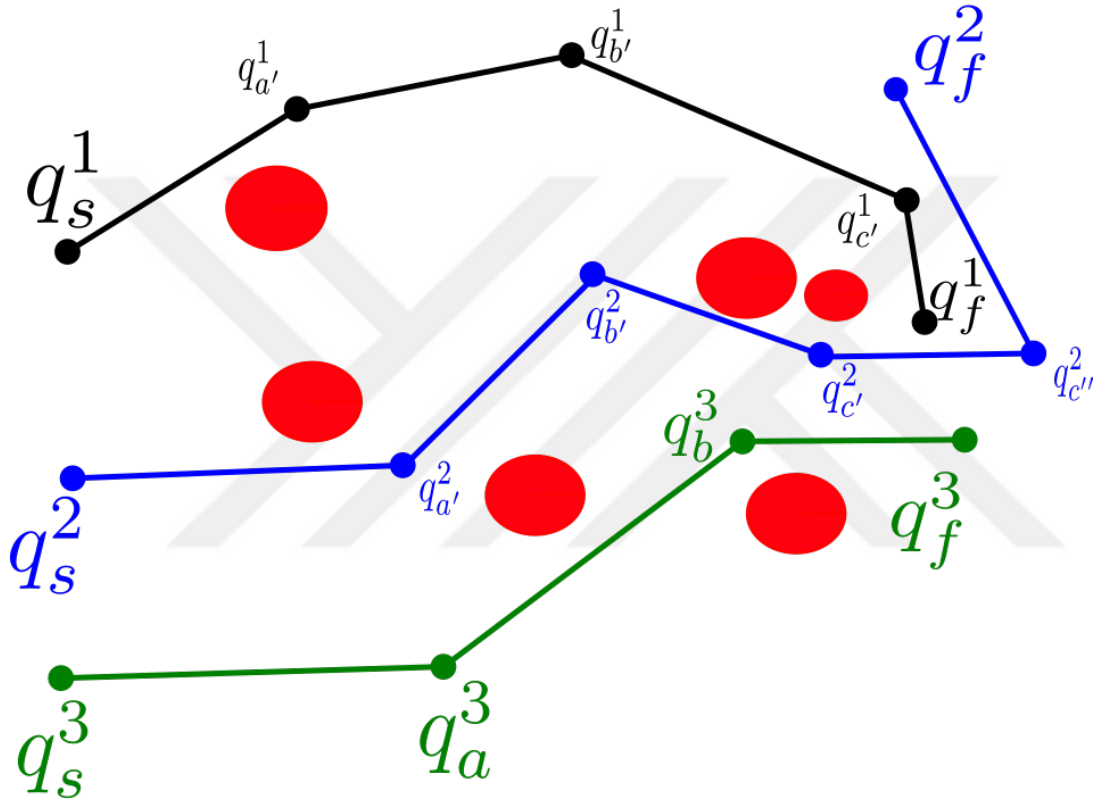
Pre-operation

$$q_k = (q_{b'}^1, q_{b'}^2, q_b^3)$$

$$T_f = \{q_s, (q_{a'}^1, q_{a'}^2, q_{a'}^3), (q_{b'}^1, q_{b'}^2, q_b^3)\}$$

$$C_k = \{r^1, r^2\}$$

$$q'_z = (q_{c'}^1, q_{c'}^2, q_f^3)$$



Post-operation

$$T_f = \{q_s, (q_{a'}^1, q_{a'}^2, q_a^3), (q_{b'}^1, q_{b'}^2, q_b^3), (q_{c'}^1, q_{c'}^2, q_f^3), (q_f^1, q_{c''}^2, q_f^3), \boxed{(q_f^1, q_f^2, q_f^3)}\}$$

$$\phi^2(q_{b'}^2) = q_{c'}^2$$

$$\phi^1(q_{b'}^1) = q_{c'}^1 \quad \phi^2(q_{c'}^2) = q_{c''}^2$$

$$\phi^1(q_{c'}^1) = q_f^1 \quad \phi^2(q_{c''}^2) = q_f^2$$

Figure 3.10: sBiRRT* robot-robot collision resolution in final step

q_z in the joint-configuration space is sampled. The robots noted in the collision set of a joint configuration-state node q_k can pick the respective parts of q_z for the sake of resolving robot-robot collisions. In Figure 3.6, r^1 and r^2 do exactly this, whereas r^3 sticks with its already valid path. The example path reconstructions in Figures 3.6, 3.8 and 3.10 are direct consequence of line 36 of Algorithm 2. Unified state-space planner also uses the greedy option introduced for single-UAV planning problem. Moreover, it is slightly different than normal BiRRT*, in which a tree is constructed from scratch. Contrarily, this planner receives the policy trees of respective robots that already contain robot-object collision-free paths. This is a significantly strong heuristic, because it enables conducting a simple robot-object collision test to p_k from lastly inserted q'_z . For instance, $q_{a'}^1$ determines its parent to be q_a^1 , just because it is not possible to directly form an edge to q_f^1 , but it is to q_a^1 . Finally, Figure 3.10 involves an illustrative example to line 36, wherein two new intermediate nodes are produced between $q_{b'}^2$ and q_f^2 , which are q_c^2 and $q_{c'}^2$. Since any combination of the robot paths in Figure 3.10 does not contain robot-robot collisions, the algorithm terminates after adding q_f vertex to its forward-growing tree T_f .

The ultimate paths in Figure 3.10 don't have any intersecting sections. This is exactly the valid output -actually only valid output- of the introduced algorithm as time-variability is not introduced to the problem. One can not determine for any robot to be on a certain section of its path just by looking at the paths described. Because of this lack of information, algorithm considers all such intersections as robot-robot collision and therefore eliminates. This obviously gives rise to unnecessarily long and thus sub-optimal paths, even when the underlying planners are optimal. This limitation can be lifted out by Velocity-Tuning method[28], which is not studied in this thesis.

3.3 Manipulator Planning

The attached 3-DOF manipulator also requires planning in order to reach the detected apples. It is essential to plan in timestamped odometry frame to refrain from localization jump issues. As ${}^M T_{odom}$ transform is fixed by construction -i.e. transform from

manipulator base to odometry sensor is fixed, it is directly possible to plan with respect to locally consistent ${}^M T_{fruit}$. This 6D pose⁵ is used to solve Inverse Kinematics (IK) problem of the attached arm. The general framework of this problem for the task at hand is:

$$q = \kappa^{-1}({}^M T_{fruit}) \quad (3.2)$$

where q represents the list of joints $q = [q_1, q_2, q_3]$. The *general* inversion problem of kinematics function κ is very difficult to be solved with an analytical solver, therefore numerical solutions come to the rescue. Consistently Eq. 3.2 can be rewritten as an optimization problem in the following form:

$$q^* = \underset{q}{\operatorname{argmin}} \left\| \log m(\kappa(q) \ominus {}^M T_{fruit}) \right\| \quad (3.3)$$

where $\ominus$ is the pose difference operator and $\log m$ is the *matrix logarithm* operator. The expression in Eq. 3.5 represents the spatial displacement between two poses, whose magnitude is aimed to be minimized. We can translate this pose math to linear algebra as follows:

$$A \ominus B = A^{-1} * B \quad (3.4)$$

Conforming to this notation, Eq. 3.5 becomes:

$$q^* = \underset{q}{\operatorname{argmin}} \left\| \log m(\kappa(q)^{-1} * {}^M T_{fruit}) \right\| \quad (3.5)$$

where $\kappa(q)$ is a 4×4 homogeneous transformation matrix, thus suitable for inversion and composition operators. Matrix logarithm is an exceptional operator in $SE(3)$, where the logarithm of a homogeneous transformation matrix represents the spatial velocity vector (twist). Conversely, exponentiation of spatial velocity vector represents the spatial displacement in the form of homogeneous transformation matrix. This well-defined duality -known as exponential mapping [14]- enables consistent bidirectional closed-form conversions of both representations, which is a crucial criterion for optimization. After computing difference transform (spatial displacement),

⁵ 3 translation and 3 rotation dimensions

it is directly possible to compute the required 6D twist, whose magnitude monotonically diminishes as both poses look more and more similar to each other. A planner such as RRTConnect [33] is used in a practical system, which produces collision-free paths in a high-dimensional cluttered workspace. It uses the underlying IK Solver both in waypoints and in the edges that connect successive waypoints when sampling. Nevertheless an already collision-free workspace -except the robot itself, so self-collisions are still respected- is assumed between the manipulator and the fruit; thus, a simple joint interpolation is employed to produce a trajectory.

`fminsearch`[34] method is used in MATLAB, a widely used nonlinear programming solver that searches for the parameter that minimizes Eq. 3.5. On the other side, KDL[35] Chain-IK Solver (C++) is employed in the Arm Planner component(see Figure 5.1), which is a substructure of `QuadManipulator`. KDL Chain-IK Solver is more or less similar to MATLAB's `fminsearch` method in terms of optimization; nevertheless, it has native support for URDF (and thus ROS) making it much more preferable.



(a) Pre-grasp pose

(b) Grasp pose

Figure 3.11: Arm Motion with IK Solution



CHAPTER 4

FRUIT DETECTION

Object detection is a settled image recognition subdomain with its flagship challenges (ILSVRC, PASCAL VOC[36]) and famous datasets (MS COCO[37], PASCAL VOC), which is mainly attacked through machine learning methods and more specifically with deep neural networks [38]. Among those, deep convolutional neural networks (CNNs)[39] became the mainstream method indisputably since they achieve much higher accuracy compared to the other methods. Historically CNNs were suffering due to the significant degradation in learning performance with the increase in the number of layers – i.e. when networks become *deeper*. On the other hand, it was shown [40] that the object detection problem could be solved with higher accuracy with deeper networks. To put it differently, higher training loss was a known phenomenon in deeper networks despite the theoretical evidence, which asserts the opposite. This dilemma remained unresolved until the introduction of ResNet [41], a deep residual learning framework for image recognition. Thereafter, ResNet became the *de facto* CNN architecture of the deeper models with intrinsically higher accuracy on image classification problems.

The task of object detection requires extracting features over different regions of the image for the sake of assigning class labels to each regressed bounding geometric shape (e.g. box) that expresses both the spatial dimension and location of the respective object, which led to the eventual development of the Region-based CNN (R-CNN) Model Family [42]. Vanilla R-CNN comprises three distinct modules to fulfill the object detection task: *region proposal*, *feature extraction* and *classification*. Region proposal module works independently from the available categories and

proposes candidate bounding boxes based on different heuristics such as *objectness* [43]. The feature extraction module employs a deep CNN to generate the features for each proposed region. Finally, the classification module assigns category labels to output feature vectors. After the tremendous success and wide adoption of R-CNN, researchers came up with Fast R-CNN[44] and Faster R-CNN [45] variants that offer much shorter training and inference times. Authors of ResNet employed the Faster R-CNN system in conjunction with ResNet50 and ResNet101 deep CNNs as feature extractors so as to compare how their neural networks perform with respect to the existing baselines on the object detection task. This setting surpassed all existing object detection baselines, further proving the utility of deep residual function learning.

Mask R-CNN[46] extends Faster R-CNN model by adding a segmentation mask prediction branch to the architecture in parallel with classification and bounding box regression branches. Interestingly it solved the object detection problem with the highest accuracy in the literature thus far by leveraging it to *instance segmentation* that requires pixel-level category classification, which is more complicated than predicting the label per bounding box.

In this study, we use Faster R-CNN and Mask R-CNN architectures accompanying ResNet50 and ResNet101 backbone networks to solve the binary apple detection problem with the highest accuracy in the literature without losing real-time inference capabilities. We denote the binary classification of the proposed image regions as apple or background with the term “binary apple detection”. To that extent, we employ the cutting-edge implementations of Faster R-CNN and Mask R-CNN under the name of Detectron2 by FAIR [47], which offers the fastest training and inference times with respect to rival/former implementations. In this setting, we first train the model on a public dataset containing a total of 800 images from a commercial fruiting-wall Scifresh apple orchard with dense foliage canopy[48] to have a desirable accuracy after hyperparameter tuning, without the explicit consideration of inference speed. Then we tweak the model by attending to several methods [49] that trade off speed over accuracy, whose output conducts inference in real-time with a maximum of 7.35 FPS on an Nvidia Tesla M40 GPU. Finally, we regain the sacrificed accuracy via *Data*

distillation [50] without increasing the complexity of the model, thus attaining most of the initially computed desirable accuracy. Such a functional flow leads to an apple detection model that surpasses existing state-of-art models with the highest accuracy and still performs in real-time.

4.1 Related Work

4.1.1 ResNet

ResNet is a residual learning framework in which the network layers fit the residual mappings of the underlying unreferenced mapping. In mathematical terms, the neural network fits $F(x) := H(x) - x$ mapping instead of directly fitting the underlying mapping $F(x) := H(x)$. The initial hypothesis of the ResNet was that it is easier to optimize the residual mapping than to optimize much more complicated original mapping[41]. They strengthen this hypothesis by speculating an extreme case where an optimal identity mapping x would lead the residual mapping to be pushed to 0, which is easier than fitting that identity mapping via a deep neural network. Finally, they empirically show that this hypothesis is correct and the phenomenal degradation problem with the increase in the number of layers is solved with residual mappings. Ultimately, this led to the invention of the deepest networks of the time, such as ResNet50/101/152. Furthermore, although ResNet152(11.3 billion FLOPs) is extremely deep, it still had lower complexity with respect to much shallower VGG-16/19(15.3/19.6 FLOPs) networks. This is because ResNet employs fewer filters compared to VGG nets. Nevertheless, it still enjoys higher accuracy via deeper network architectures thanks to the residual learning framework.

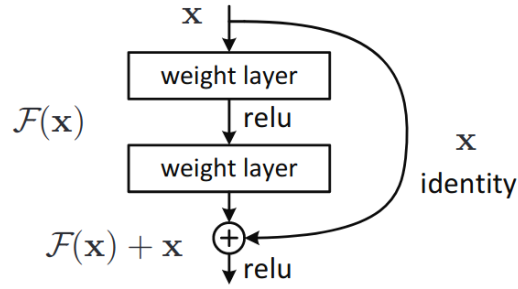


Figure 4.1: Residual learning: a building block. Reprinted from *Deep Residual Learning for Image Recognition* by Kaiming He et al. 2016, CVPR

Figure 4.1 illustrates the shortcut connection within the residual block, which actualizes the identity mapping. Those mappings are added to layer outputs, leading to learning of residual mapping without any extra parameter or computational overhead. This simple yet strong component enabled the achievements of ResNet. Formally, building blocks are generally defined as:

$$y = F(x, \{W_i\}) + W_s x \quad (4.1)$$

Equation 4.1 defines the input and output vectors of a residual block as illustrated in Figures 4.1 and 4.2. W_s is only used when the dimensions of F and x mismatch, which happens when altering the number of channels between input/output of the conv. layer.

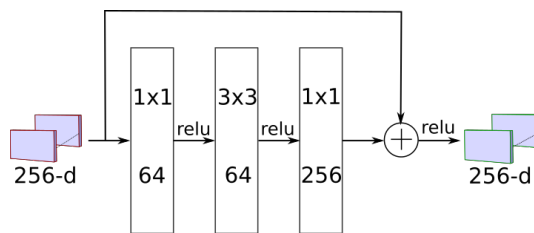


Figure 4.2: Bottleneck building block

ResNet consists of an initial basic stem block and consecutive stages that contain sets of bottleneck blocks with varying cardinalities. For instance, ResNet50 uses a total of 16 bottleneck blocks in Conv2/3/4/5 stages. Since the bottleneck block in Figure 4.2 consists of 3 layers, this makes a total of 48 layers in the respective convolution

stages. The basic stem comprises a $K=7$, $S=2$ filtering followed by a $K=3$, $S=2$ max-pooling operator that sums the number of layers up to 50. The sole purpose of the bottleneck block is to reduce the training time by leaving the inner 3×3 conv. layer to operate on smaller input/output dimensions by compressing and expanding the number of filters/channels used in the volumetric convolution operations.

4.1.2 R-CNN

As mentioned, object detection requires a model from R-CNN model family. Therefore ResNet authors used Faster R-CNN framework to compute their object detection scores and achieved the best results of the time. Before delving into Faster R-CNN details, it is crucial to iterate over vanilla R-CNN, which has a prolonged test detection time of 0.077 FPS on a GPU[42] due to the excessive number (~ 2000) of candidate regions, which need to be sequentially piped into deep CNN for feature extraction. This architecture leads to a multi-stage learning pipeline that compromises both speed and accuracy in training due to independence between stages — i.e. the region proposal module does not care about the evaluation done in the classification stage; hence, the proposals do not improve over time.

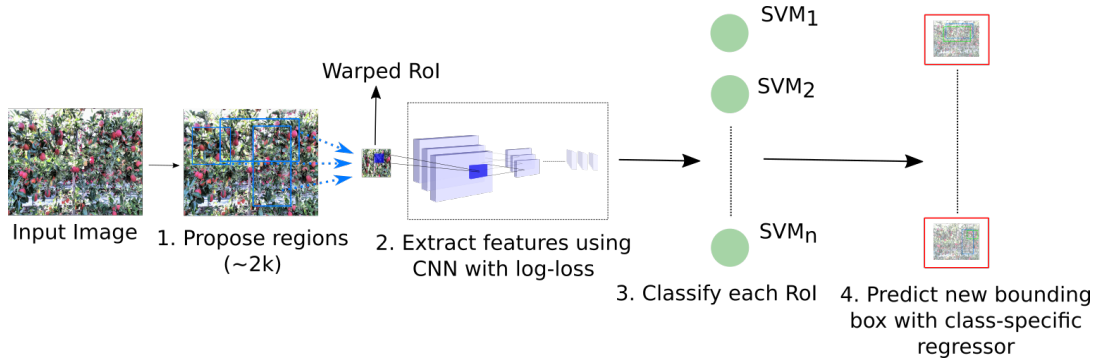


Figure 4.3: R-CNN Architecture

R-CNN replaces softmax classifiers from the final layer of CNN with class-specific SVMs, each of which is fitted to CNN features according to their ground-truth category. This design choice is purely empirical in the sense that the negative examples used in the fine-tuning process were randomly selected and thus caused the overall

mAP score to be 4 points lower when compared to the prediction score of the set of SVM classifiers trained for *hard negatives*. Nevertheless, authors admit in Appendix B of [42] that the hard negative mining method [51] used in SVMs could be compensated with other tricks in fine-tuning process for the sake of using softmax classifiers directly in the inference stage without suffering detection performance, which would also improve overall training time. Figure 4.3 represents the multi-stage training pipeline used in R-CNN. Initially, a region proposal algorithm offers approximately 2000 regions in the form of bounding boxes. Subsequently, CNN extracts features from each candidate region-of-interest (RoI) and classifies them in the final softmax layer during fine-tuning and with a set of SVMs in training. Subsequently, RoI-extracted CNN feature vectors are given as input to the class-specific bounding box regressor of the class with the highest prediction score from the SVMs of the previous stage to reduce the object localization error.

4.1.3 Fast R-CNN

It is apparent that the multi-stage training pipeline, naïve region proposal algorithm and unconditional CNN forward pass applied on each candidate RoI hinders both the training and inference performance, effectively rendering R-CNN unavailable for practical usage [44]. Those significant drawbacks are eliminated and improved with Fast R-CNN variant via several innovations. The most notable contribution of this variant is a single-stage training scheme that uses multi-task loss, which is fundamentally different and faster than the multi-stage pipeline of the vanilla R-CNN. The second important innovation is the deferred feature extraction via *RoI pooling layer* (RoIPool). Contrarily to vanilla R-CNN, this variant computes the feature map of the whole image in a single forward pass using a deep CNN (e.g. ResNet) and then extracts a fixed-length feature vector from that feature map for each candidate region using the custom RoIPool followed by a sequence of FC layers. These two improvements alone provide significant drops in training ($9\times$) and inference ($7\times$ plain; $10\times$ with truncated SVD[52]) times; nevertheless, there are few other things introduced to improve detection quality and time & space complexity, which will not be discussed in here.

RoIPool converts the whole-image feature map by applying max pooling to each independent channel on each particular RoI window of any size $h \times w$. The output of this layer is of fixed-length for any RoI with an $H \times W$ spatial extent, where H and W are layer hyper-parameters. These fixed-length feature vectors are fed into a sequence of FC layers, which finally bifurcates into two sibling output layers: softmax classification and bounding box regression that respectively outputs probability estimates for $K + 1$ (incl. *background*) and 4-tuples for K classes.

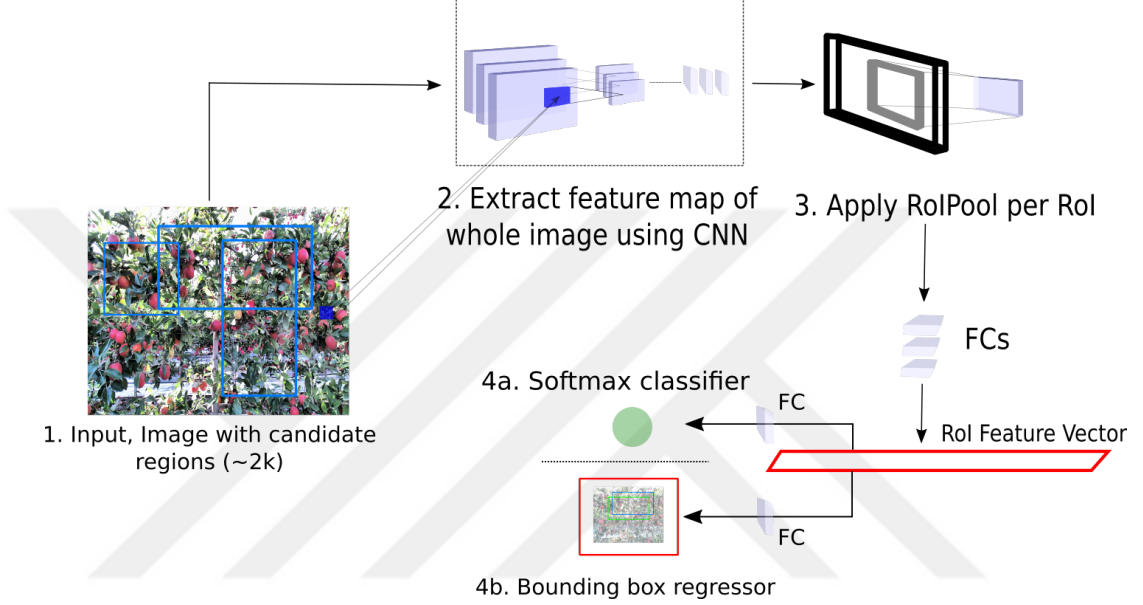


Figure 4.4: Fast R-CNN Architecture

The losses of output layers in Figure 4.4 are combined in a multi-task loss formulation so that the model would simultaneously learn to predict better bounding boxes for improved localization and higher classification success rate for the overall object detection problem.

$$\begin{aligned}
 L_{det} &= L(p, u, t^u, v) = \alpha L_{cls}(p, u) + \beta [u \geq 1] L_{loc}(t^u, v), \quad \text{where} \\
 L_{cls}(p, u) &= -\log p_u \\
 L_{loc}(t^u, v) &= \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L1}(t_i^u - v_i) \\
 \text{smooth}_{L1}(x) &= \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases}
 \end{aligned} \tag{4.2}$$

Equation 4.2 defines all the components of the multi-task loss. L_{cls} is the log loss for true class u with respect to the discrete probability distribution $p = (p_0, \dots, p_K)$. L_{loc} is the representation of localization loss, which computes the mismatch between the ground truth bounding box 4-tuple $v = (v_x, v_y, v_w, v_h)$ ¹ and predicted tuple $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$. $[u \geq 1]$ is the Iverson bracket indicator that evaluates to 1 when the conditional statement holds and 0 otherwise, which is the compact representation of conditional loss function. The rationale behind $smooth_{L1}$ loss is the fact that the unbounded regression targets could explode gradients in an L_2 loss setting, requiring more careful tuning of learning rates. The hyper-parameters α and β are weighting factors that allow the adjustment of importance on the particular tasks; however, they are kept as $\alpha = \beta = 1$ in all experiments of [44].

Multi-task loss training not only reduces the training & inference time drastically due to its single-stage nature, but it also improves the accuracy since the tasks are not entirely independent, i.e. they influence each other through a shared representation. The gradients of multi-task loss L are back-propagated to the internal layers of deep CNN through the RoIPool layer, allowing the bounding box regression loss to affect neurons in the early layers of CNN. This is a significant feature that vanilla R-CNN was unable to provide. Consequently, the shared features are empirically proven to be better learned with multi-task loss. Although the learning-free region proposals still create a significant bottleneck in the inference performance, leaving the model real-time incapable.

Lastly, the one-vs-rest SVM classifiers used in vanilla R-CNN are removed for the sake of using the softmax classifiers from the one-shot training session, which is another proof of single-stage learning. Interestingly softmax classifiers slightly outperformed SVM classifiers with the same hyper-parameters as those used in vanilla R-CNN. This improvement can be attributed to the accuracy gain due to the multi-task loss formulation described above.

¹ It is redundant to provide u superscript here as the ground-truth bounding box is uniquely defined for class u alone.

4.1.4 Faster R-CNN

The authors further improve the Fast R-CNN version in this variant by introducing Region Proposal Networks (RPNs)[45]. Region proposals are included in the learning process with this contribution, leading to better proposals through acting as an *attention* mechanism, which tells the subsequent network where to *look at* and thus higher accuracy scores for the downstream Fast R-CNN stage in the object detection task. Furthermore, some of the convolutional layers are shared between two semantically distinct networks qualifying the name of the study by rendering already shallow RPN networks almost cost-free. Depending on the training scheme, Faster R-CNN could be considered a 2-stage learning model where RPN constitutes the first stage and Fast R-CNN substructure constitutes the second. This study considers *approximately joint training* [45], which merges RPN and Fast R-CNN networks into one. In this setting, each SGD forward pass generates region proposals from RPN, which are considered as pre-computed proposals in the downstream object detection network. Consistently, back-propagation takes both L_{RPN} and L_{det} losses into account for shared layers. Nevertheless, those signals are approximate in the sense that Fast R-CNN loss does not compute the gradient for RoI proposals in the partial derivatives as they are considered fixed in the downstream network. In contrast, in reality, they are generated through RPN and have valid gradients. It is reported that this training scheme reduces the training time by 25-50% with respect to its rival – alternating training [44].

The aforementioned convolutional layer sharing between RPN and object detection networks, which eliminates the previously described region proposal bottleneck, is enabled by the introduction of *Pyramid of Anchors*. Typical region proposal algorithms used pyramid of images or pyramid of features to create multiple scales and aspect ratios in order to generate *anchors*, any of which would prevent the critical feature sharing of Faster R-CNN. On the opposite, using multi-scale anchors as regression references allows re-utilization of single-scale whole-image features extracted through the deep CNN, which is already exploited by Fast R-CNN object detector, without any extra costs.

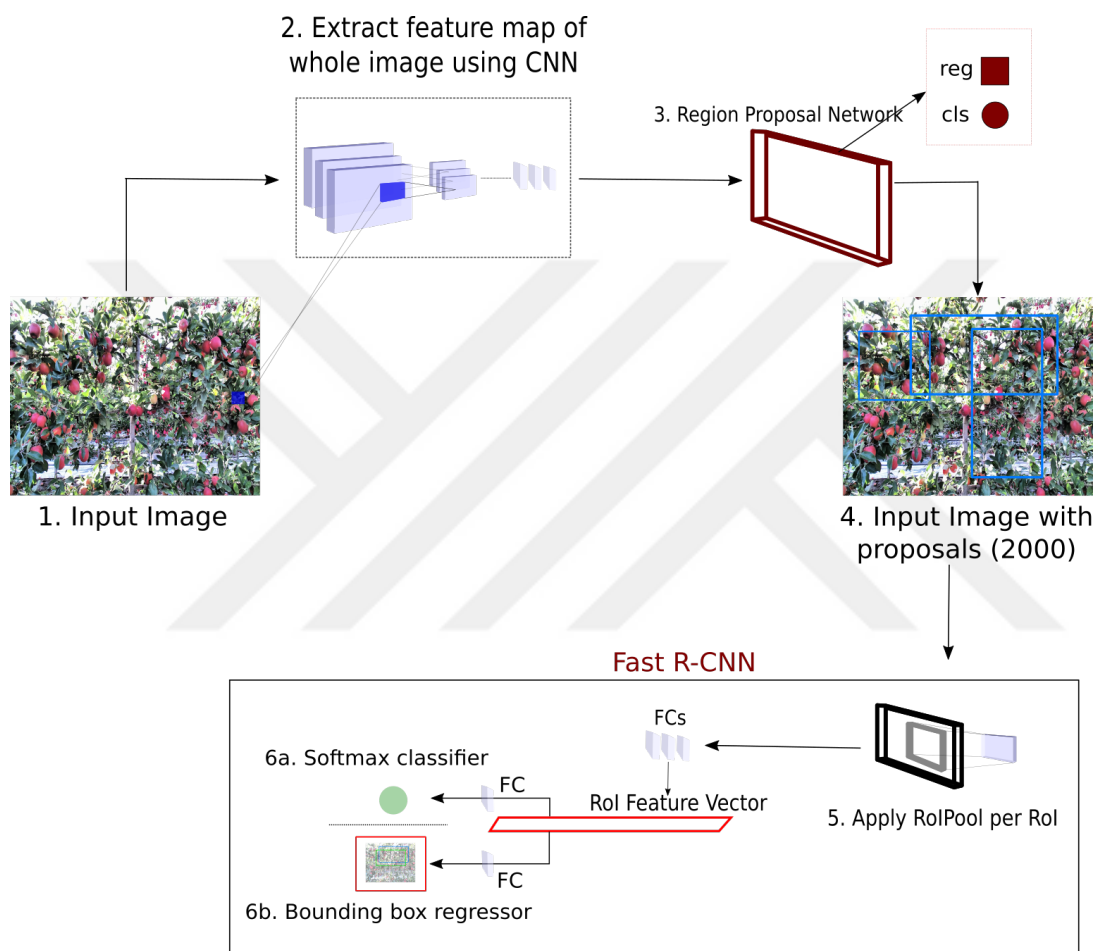


Figure 4.5: Faster R-CNN Architecture

RPN simultaneously predicts bounding boxes and objectness scores on each anchor that is generated according to a single-scale whole-image feature map using multiple scales and aspect ratios. In order to learn better region proposals, RPN loss function distinguishes between positive and negative anchors and ignores non-negative and non-positive anchors during training. An anchor is assigned the positive label if it meets one of the following criteria based on its Intersection-over-Union (IoU) ratios:

- it has the highest IoU with respect to a ground-truth bounding box,
- it has an IoU overlap that is higher than 0.7 with any ground-truth bounding box,

Similarly, negative anchors are determined when a non-positive anchor has an IoU ratio that is lower than 0.3 for all ground-truth boxes. This distinction is crucial in the designation of the loss function so that the learning would not fail. RPN employs a multi-task loss function as well using the same primitives from Equation 4.2, but with different hyper-parameters.

$$\begin{aligned}
 L_{RPN} = L(\{p_i\}, \{t_i\}) &= \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, o_i) \\
 &+ \lambda \frac{1}{N_{reg}} \sum_i o_i L_{reg}(t_i, v_i), \quad \text{where} \\
 o_i &= \begin{cases} 1, & \text{if } i \in \text{positive anchors} \\ 0, & \text{otherwise} \end{cases} \\
 L_{reg}(t_i, v_i) &= \text{smooth}_{L1}(t_i - v_i)
 \end{aligned} \tag{4.3}$$

Equation 4.3 illustrates the multi-task loss used in RPN training. N_{cls} , N_{reg} and λ are set to 256, 2400 and 10 respectively in [45]. λ is particularly set as 10 in order to balance *cls* and *reg* terms so that the loss function has approximately equal weight with respect to errors received from those output layers, which are visible in Figure 4.6. o_i represents the ground-truth value of whether the anchor i is an object or not based on its negative/positive/none label, whereas p_i represents the objectness prediction.

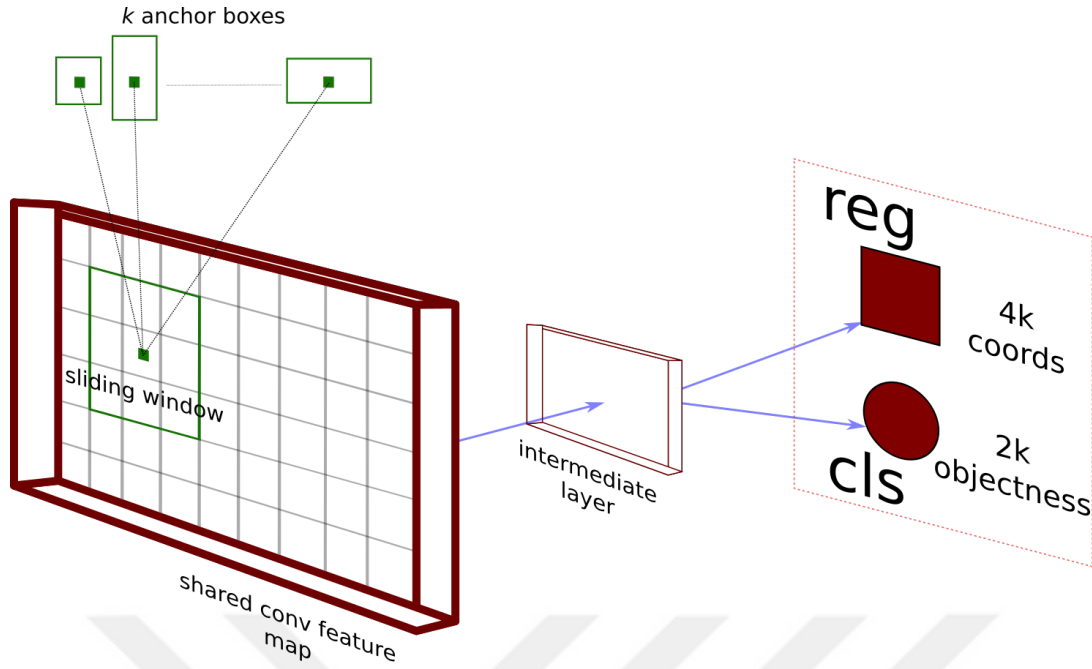


Figure 4.6: Region Proposal Network

k in Figure 4.6 represents the maximum possible proposals, which is set to 2000 in Faster R-CNN training and to different values (300, 1000, 2000) in testing for comparisons. *reg* layer outputs $4k$ values for encoding the coordinates of k bounding boxes, whereas the *cls* layer outputs $2k$ scores as the probability estimations of objectness of the particular anchor.

Consequently, RPN, which shares some of its constituent conv. layers from downstream object detection network, costs an additional 10ms per image in time complexity during the inference. This improvement leads to 198 ms inference time per image with VGG-16 backbone, i.e. 5 FPS [45]. As the authors claimed, this is a near real-time performance.

4.1.5 Mask R-CNN

Mask R-CNN is the intuitive extension of Faster R-CNN, which adds a mask prediction branch to the architecture for the segmentation task. Unsurprisingly, the mask prediction loss included in multi-task loss also improves the object detection accu-

racy. This is the similar phenomenon observed in the Fast R-CNN variant, wherein the RoI bounding box regression loss is injected into the overall loss function.

The core innovation of [46] is the novel RoIAlign layer, which fixes the misalignments introduced through coarse spatial quantizations via RoIPool – i.e. outcomes of divisions with strides rounded to nearest integers and discretely sized bins were causing approximations in the original continuous state-space, effectively breaking the pixel-to-pixel alignments between network inputs and outputs. RoIAlign straightforwardly fixes this issue. It does not use any quantization; instead, it computes the exact values of the input features using bilinear interpolation [53] at four regularly sampled locations in each RoI bin. Outputs from the interpolation procedure are aggregated with max or average operators to produce the exact features.

4.1.6 FPN

Feature Pyramid Network (FPN) [54] is a type of network that produces feature maps in different scales and aspect ratios. These feature maps are produced in diminishing sizes, which can be considered as the *pyramid of features*. FPN uses a component called lateral convolution that takes features from different stages (e.g. res3) of underlying CNN backbone and manipulates them all to have 256-d channel uniformly. The output features from the latter stages are then fed respectively into output convolution layers to constitute one of the FPN outputs and to upsamplers so as to feed the previous stages. In the end, P2, P3, P4 and P5 output convolution layers end up with features respectively downscaled with a factor of 4,8,16 and 32.

Figure 4.7 represents the P2, P3, P4 and P5 feature maps of the test image shown in Figure 4.12 from the 0th channel in gray-scale format. The receptive fields of each layer become coarser to the tip of the pyramid, capturing fewer and possibly different details of the input image.

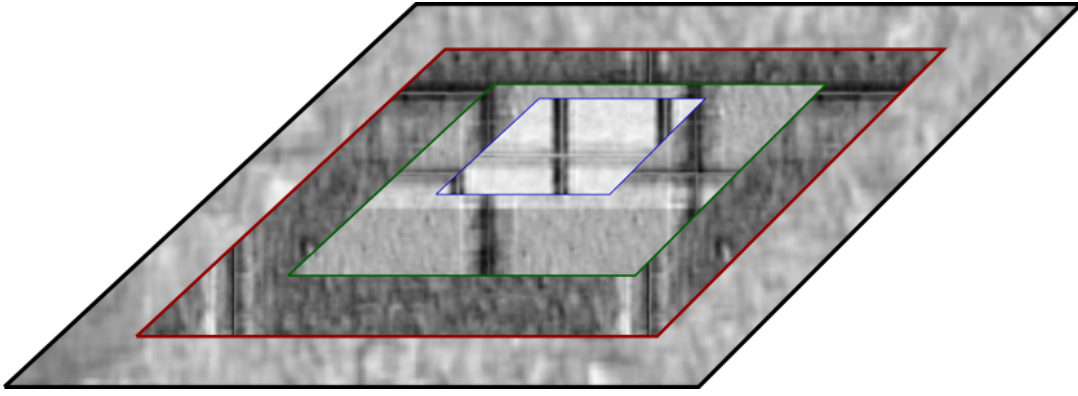


Figure 4.7: FPN's Pyramid of Features

4.1.7 Apple Detection

[55] uses Faster R-CNN with VGG-16 backbone to produce a general-purpose fruit detection model. To that end they fuse RGB and Near Infrared (NIR) sensor inputs in the training stage to improve the overall accuracy. They use a relatively smaller dataset on apples consisting of 64 images, on which they report an F1 score of 0.938. [56] solves a slightly different problem than attacked in this paper. They construct a ternary object detection problem in which there are two different classes of apples and one background class. They employ the cutting-edge Mask R-CNN[46] architecture using ResNet101-FPN backbone with a feature suppression end, which results in a large F1 score improvement of 5% on their own baselines. Eventually, the F1 score is reported as 0.905 and test-detection speed is found to be 4 FPS. Finally, [48] coins the same binary apple detection problem of this study. Annotated 800-images dataset created in the scope of [48] is used in this work. They use Faster R-CNN architecture with VGG-16 and ZFNet backbones. As they have also collected depth information, they have applied depth filtering of 1.2 meters in the pre-processing stage to filter out the apples that are not in a robot's workspace. They report 0.839 AP and 5.52 FPS with VGG-16, whereas ZFNet produces 0.805 AP and 8 FPS on an Nvidia GeForce TITAN XP 6GB GPU card along with Intel Xeon E5-1650 6-core CPU. We have manually adjusted this filtered dataset to its unfiltered version and used it both in evaluation and training.

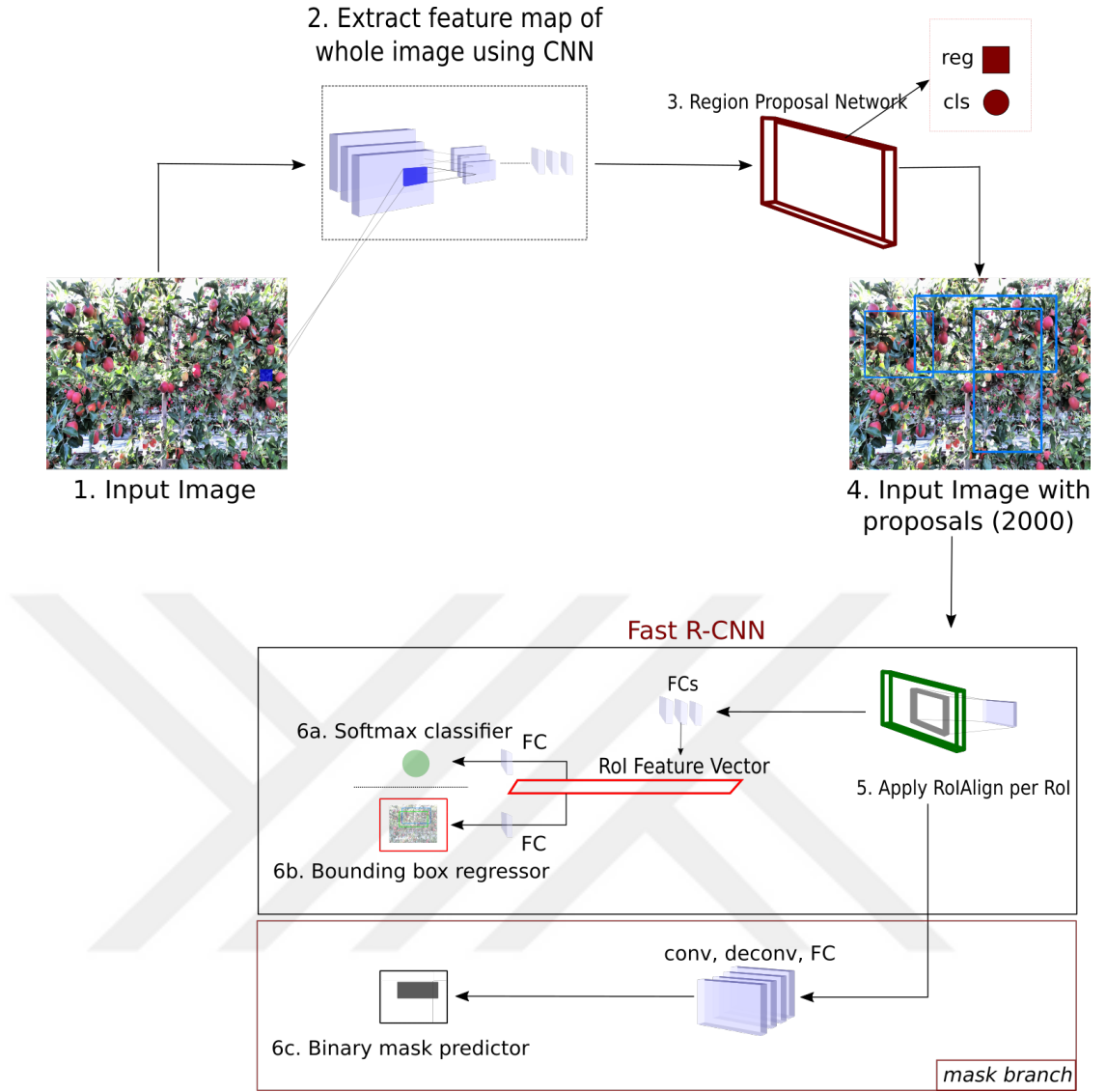


Figure 4.8: Mask R-CNN Architecture

4.2 Method

We devised a binary object detection problem in order to localize the apples in a particular image. To that end, there are an *apple* and a “catch-all” background classes. ResNet50/101-FPN pre-trained models are employed as the backbone architecture for comparison. FAIR’s Detectron2[47], which provides the extendible open-source architecture in Figure 4.8, is used to conduct training and inference on the existing 800-image wall-fruited apple orchard dataset.

$$\begin{aligned}
L_{mrcnn} &= L_{det} + \lambda[u \geq 1]L_{mask}, \quad \text{where} \\
L_{mask}(d_{m \times m}^u, n_{m \times m}) &= - \sum_i^{m \times m} n_i \log(s(d_i^u)) \\
&\quad + (1 - n_i) \log(1 - s(d_i^u))
\end{aligned} \tag{4.4}$$

Eq. 4.4 illustrates the overall loss function of Mask R-CNN and the particular mask branch loss used in training. $d_{m \times m}^u$ represents the mask prediction on a particular RoI. $n_{m \times m}$ is the ground-truth mask of class u generated in pre-processing phase. $s(\cdot)$ is the sigmoid function, applied on a per-pixel basis to the predicted mask that returns the binary classification for the particular class u . In the end, mask branch loss is the average binary cross-entropy loss, added with the Iverson bracket indicator.

4.2.1 Dataset Preparation

We started the study first by inspecting the quality of the annotations of the entire dataset. Since we will not use depth information to train and evaluate the model, it is indeed reasonable to coin the following question: *Is it meaningful to filter out distant apples?* The answer to this question is mostly yes, but with a caveat. It is yes because detecting distant fruits is of no use for a robotic harvesting system as it would not be able to pick them since they would be located out of reach of its manipulator. One could manually filter out distant fruits even in the absence of depth information by eliminating out the ones that are smaller than a certain size for the sake of sacrificing outlier fruits that have grown prematurely. In the existence of depth sensory input, this procedure can be automated and the error rate reduces to the noise in the sensory system. However, one big caveat may hinder the performance of learning systems, primarily when the depth information is not used *during* training but only used in pre/post-filtering stages. Since the typical depth cameras send their rays from a point source (e.g. from the robot's head), the fruits that are close to the edges of the field of view are naturally located more distant than the ones that are positioned in the center, even when they are almost next to each other. In this setting, one of the side-to-side apples might be categorized as *apple* in annotation, whereas the other one can be categorized as background or *catch-all* class.



Figure 4.9: Filtered Annotation Example

Figure 4.9 illustrates this clearly. It is crucial for an object detection model that uses only image data for training since it will not have a structured concept of a 3-dimensional spatial world². Apples bounded with red rectangles denote the objects in the image that are annotated as *not-apples*. Failing to categorize them properly as *apples* would cause the model weights to stumble unless depth data is also used in training since very similar features from the image portions located more to the center were labeled as apples. This inevitably hinders the training and the resultant model would be slightly less precise as it tweaks its internal structure based on only the 2D image plane inputs, hence simply unable to capture certain 3D features, which in return leads to a model that has an increased rate of false positives. We summarize all these observations such that the training input modalities and annotations need to be consistent. [57]

$$AP = \int_0^1 p(r)dr, \text{ where } p(r) \text{ is precision-recall curve.} \quad (4.5)$$

In order to test the abovementioned hypothesis, we have manually reannotated the first 80 images from the 800-images filtered dataset to its unfiltered version - that is,

² However it may learn to some extent with specialized loss functions, which is out of scope in this study.

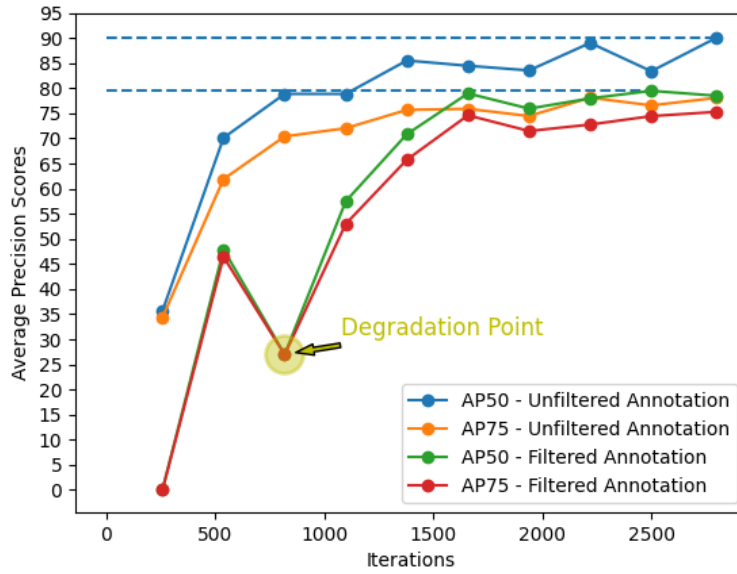


Figure 4.10: AP scores of the models trained with filtered and unfiltered annotations

whose annotation is not pruned according to depth data - so as to compare the average precision scores of models trained on both dataset versions for 50 epochs. Figure 4.10 represents the AP50 and AP75 scores on the validation sets of both versions, with a split of 70% and 30% on 80-image dataset. We use iterations to represent the training progress in figures of all subsequent experiments. The number of maximum iterations during training is simply the multiplication of the training subset size and number of epochs.

Maximum AP50 scores of the models trained on filtered and unfiltered annotations have a significant gap in favor of the unfiltered version, with an improvement of 10.0% in AP50 metric. Despite being relatively smaller, the difference between AP75 scores of both models further approve our hypothesis: It is important to provide annotations that match the modalities used in training – unless those other sensory inputs have a learnable mapping to the ones used in training. Moreover, there is a degradation point in the model, which is trained on the filtered dataset at ~1000 iterations, which illustrates the fact that the model falters on its predictions with respect to the image inputs in the early stages. Subsequently, it learns to distinguish between distant and closer fruits more successfully; however, its performance is suboptimal from

a pure object detection perspective. Would the learning procedure be accompanied with the depth information, the plots in Figure 4.10 would be reverted in favor of the model trained on the filtered dataset.

4.2.2 Training

After fixing the annotations, we have split the dataset into training, validation and test datasets with the respective percentages of 70%, 15% and 15%. Due to the limitations in the implementation of Detectron2, only three hyper-parameters are considered: learning rate, RoI bounding box regression loss weight and RPN bounding box regression loss weight, of which the latter two are visible respectively in Figures 4.4 and 4.6. For the hyper-parameter search problem, Bayesian Optimization[58] is considered the most suitable method due to the non-linear nature of the target function – validation loss. The validation loss is negated since the particular implementation is only able to maximize the target function. The range of search is:

- Learning-rate: (0.001, 0.01)
- RoI bounding box regression loss weight (β in Eq. 4.2): (0.0, 1.0)
- RPN bounding box regression loss weight (λ in Eq. 4.3): (0.0, 1.0)

Table 4.1: AP, F1 and FPS scores of Mask R-CNN on Apple detection

Parameters	Backbone	Epochs	IoU	th_score	AP	F1	FPS ³
Default	ResNet-50-FPN	50	0.75	0.95	0.92	0.929	2.31
Tuned	ResNet-50-FPN	27	0.75	0.95	0.96	0.967	2.31
Default	ResNet-101-FPN	50	0.75	0.95	0.918	0.938	1.89
Tuned	ResNet-101-FPN	27	0.75	0.95	0.95	0.96	1.89

In the Bayesian optimization procedure, 10 initial random points -i.e. 3-tuples of hyperparameters- are explicitly requested from the solver to explore the 3D state

³On Nvidia GTX 1050

space, which are then fed into the training scheduled to run for 3 epochs. At the end of the training, validation loss is computed through evaluating the validation set and returned to the optimization procedure after being negated. After that, the most promising result is chosen for stricter optimization and parameter bounds are narrowed down to $(0.004, 0.006)$, $(0.05, 0.2)$ and $(0.05, 0.2)$ respectively. After that, exploration is discouraged by setting the $kappa = 0.5$ while using upper confidence bound acquisition function, which is a principal function used to trade-off exploration vs exploitation.

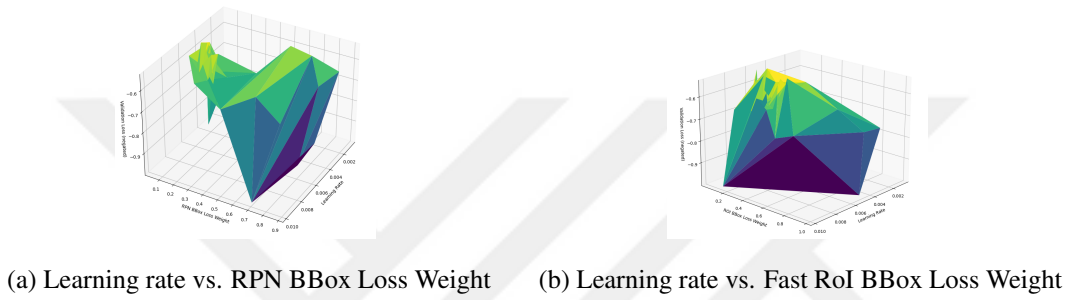


Figure 4.11: Hyperparameters vs. Validation Loss

In both plots of Figure 4.11a, the global maximum is located at $LR = 0.0055$, RoI BBox Loss Weight = 0.05 and RPN BBox Loss Weight = 0.162. After settling down on hyper-parameters, we have conducted 50-epoch training to figure out where more training becomes negligibly useful.

Figure 4.12 shows that the validation loss starts to plateau at ~ 18000 iterations, roughly 27 epochs. Since the models are saved at every 2000 iterations, the corresponding weights are directly used for evaluation.

Table 4.1 represents the evaluation scores. The model using ResNet-50-FPN as backbone deep CNN with tuned hyper-parameters outperforms the other versions. Nevertheless, the FPS score of 2.31 is still insufficient for real-time usage. Inference time needs to be reduced with additional measures.

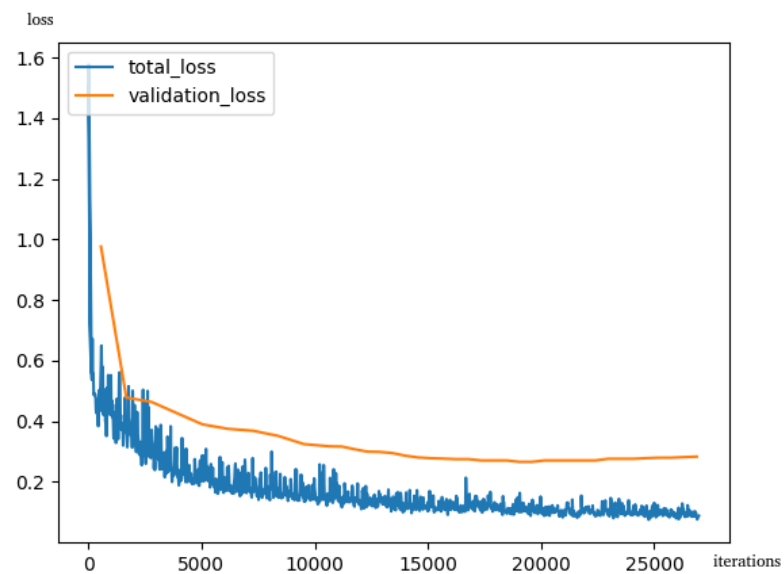


Figure 4.12: Training vs Validation Losses

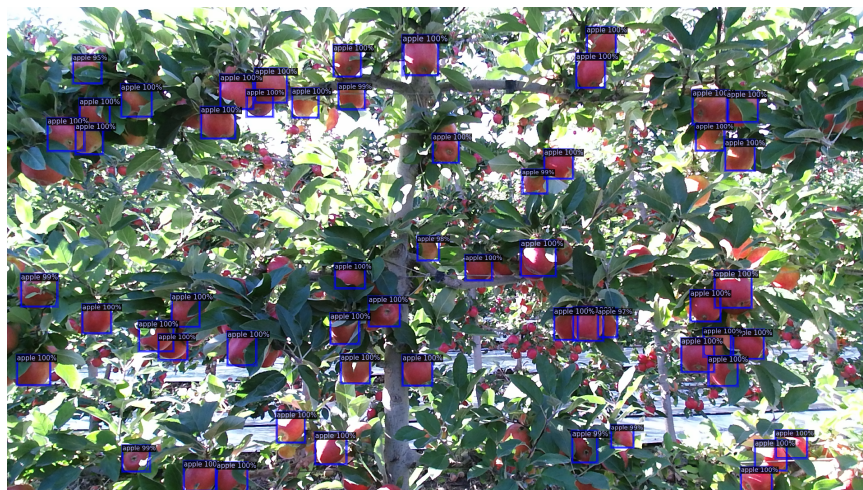


Figure 4.13: Apple detections on a test-set image

4.2.3 Speed vs. Accuracy Trade-off

Table 4.2: Trade-off between accuracy and speed of different parameter combinations

Experiment	Meta-Architecture	Input Size	Region Proposals	AP	F1	FPS ^a
1	MR-CNN ^b	1200	1000	0.9517	0.96	2.72 / 7.2
2	MR-CNN	1100	1000	0.908	0.931	3.04 / 8.0
3	MR-CNN	1200	50	0.824	0.831	3.36 / 8.9
4	MR-CNN	1100	50	0.742	0.76	3.9 / 10.3
5	FrR-CNN ^c	1200	1000	0.918	0.934	3.64 / 9.6
6	FrR-CNN	1100	1000	0.863	0.88	4.3 / 11.3
7	FrR-CNN	1200	50	0.764	0.799	4.6 / 12
8	FrR-CNN	1100	50	0.692	0.702	5.6 / 14.9

^a GTX 1050 / Tesla M40

^b Mask R-CNN

^c Faster R-CNN

A number of speed vs. accuracy trade-off methods are enumerated in [49]. Among those, we have decided to alter the number of region proposals and input image size so as to increase the inference rate. After the extensive analyses conducted on a number of meta-architecture, feature extractor and hyper-parameters combinations, it is further stated in [49] that there exists an optimal selection of those parameters, which achieves the best balance between speed and accuracy. This selection is called *sweet spot*, which is Faster R-CNN with ResNet-50-FPN limited to 50 region proposals. In this study, we use Mask R-CNN, a natural extension of Faster R-CNN, which employs an additional mask branch to improve accuracy. Nevertheless, this additional branch introduces a certain amount of slowdown in inference, which might hinder its usefulness in real-time. To that end, we compare Mask R-CNN with respect to Faster R-CNN through several hyper-parameter combinations. We have used a standard desktop computer with 24 GB RAM, Intel i5-7300HQ processor, and an Nvidia GeForce GTX 1050 GPU card in all experiments. We have calculated a constant multiplier of 2.65 between host and target -namely the machine used in [46]- machines based on the achieved and reported FPS values respectively on Mask R-CNN meta-architecture with ResNet-101 backbone, which is using 1000 region proposals

at inference and original image sizes. Finally to note, we opt for ResNet-50-FPN backbone and IoU score of 0.75 in all experiments.

Table 4.2 contains 8 experiments conducted with several combinations of parameters. The optimal result is observed in *Experiment 5*, where the AP and F1 scores are substantially over 0.9 and detection time is almost 10 FPS. Nevertheless, it should be noted that the Mask R-CNN architecture with the same parameters (*Experiment 1*) has a substantial improvement of $\sim 3\%$ both in AP and F1 scores. The second-best outcome is observed in *Experiment 2*, which has a slightly lower input image size with respect to *Experiment 5*. Despite that, it still achieves a lower inference speed of 8 FPS due to added computational overhead of the mask branch. Lastly, *Experiment 6* looks promising, although it has relatively lower accuracy scores with respect to the combinations above since it significantly surpasses both the accuracy scores and detection speeds obtained in [48]. Unfortunately, we can not perform an apple-to-apple comparison for [56] on inference speed since they do not expose their test-bench configurations but instead state that “a standard desktop computer” was used.

Consequently, it needs to be acknowledged that the major distinction that leads to significant speedups in inference compared to former studies is the novel Detectron2[47] framework. Having said this, it is still non-trivial to adjust certain hyper-parameters (e.g. input image size) to trade-off between speed and accuracy depending on the application.

4.2.4 Data Distillation

Model ensemble [59] is a simple idea to improve the accuracy of a target problem by employing different models and then taking the weighted average of their predictions. Following this idea, Caruana et al. illustrated [60] the possibility of simpler models that can be run on low-compute devices -e.g. on the edge- could distill the knowledge from a number of cumbersome models that are incapable of providing real-time inference. After that, model compression (or *knowledge distillation* in a fancier way) has become a hot research topic, which seeks means of emulating the accuracy per-

formance of the heavy neural networks in faster ways. Essentially there is an RMSE loss in model compression technique that tries to minimize the loss in logits space between teacher models and student models, where the predictions are produced with softmax output layers. Hinton et al. generalizes upon this technique by making the softmax layer *softer* through introducing T hyperparameter [61].

Knowledge distillation techniques thus far focus only on ensemble models so as to distill the knowledge into a simpler model. Nonetheless, [50] introduces a novel idea on ensemble predictions done with respect to a set of different transformations applied on the same input by a teacher model. This novelty cuts a lot of training time as there is only one teacher model, contrarily to the model ensemble, where many cumbersome models are needed. This technique is a natural extension of “self-training”, where the unlabeled data is used to produce predictions that are later used to update the model. More importantly, there is no necessity of constructing a minimization pipeline on *distillation loss* to distill knowledge. Finally, data distillation does not use the concept of soft labels, which are probability vectors of existing classes, but instead uses hard categorical labels via non-maximum suppression.

In this study, we use the aforementioned data distillation technique to regain some amount -if not all- of accuracy that was traded off in Section 4.2.3. In order to produce auto-annotations on unlabeled data, we needed to ensemble the bounding box predictions. To that end, we have created a bounding box ensemble algorithm based on [62], which is shown in Algorithm 3.

Algorithm 3 is very similar to what is listed in [62] in spite of a subtle difference in **match_score** and **box_merge** functions. The sum of distances between centers of boxes and intersection area of boxes is stated to be used as the scoring logic in [62], which is unintuitive since the distance between centers of boxes has an inverted relation with respect to the intersection area. Therefore we use the gated pixel-space distance between centers of boxes as the scoring logic, which returns a value higher than t if the pixel-space distance is higher than 30 or IoU between the pair of boxes is lower than 0.5. Otherwise, it returns the actual pixel-space distance between centers.

Algorithm 3 Bounding-Box Ensemble

```
1: procedure ENSEMBLE_BOXES( $B_t$ )
2:    $B \leftarrow \bigcup_t B_t$ 
3:   while true do
4:      $(b_1^*, b_2^*) \leftarrow \operatorname{argmin}_{b_1 \neq b_2 \in B} \operatorname{match\_score}(b_1, b_2)$ 
5:     if  $\operatorname{match\_score}(b_1^*, b_2^*) > t$  then
6:       return  $B$ 
7:     else
8:        $B \leftarrow B \setminus \{b_1^*, b_2^*\} \cup \operatorname{box\_merge}(b_1^*, b_2^*)$ 
9:     end if
10:  end while
11: end procedure
12: 

---


```

Moreover, we take the average of b_1^* and b_2^* in **box_merge** method, whereas [62] applies NMS on them. We set t to 30, which is an application-specific hyperparameter that depends on the use-case. Finally, we reject bounding boxes with prediction scores lower than 0.85 to eliminate possible false positives as suggested in [50].

This paper employs scaling, horizontal flipping and intensity transformations on unlabeled input images to produce a set of regressed bounding boxes. We have applied the cartesian product of all transformations – namely 9 scales ranging between [400-1300], 6 different intensities ranging between [0-1] and flip or no-flip. These lead to a total of 108 predictions on transformed inputs. Then we ensemble these bounding box regressions for each image using Algorithm 3. After ensemble, we produce the annotations of unlabeled images based on the predictions and call them auto-annotated images.

We devised the tuned Mask R-CNN architecture with ResNet-50-FPN backbone from Table 4.1 as the teacher model, who conducted predictions on the several transformations of unlabeled data, which are then assembled by Algorithm 3 to create auto-annotations. We then continued with a retraining session from scratch based on the union set of the initial dataset and the auto-annotated images dataset, which respectively consists of 800 and 224 images, with 100 epochs to account for additional images. Table 4.3⁴ represents the accuracy scores of the models after retraining with the union of labeled and auto-labeled datasets. There is a visible ~ 0.5 -1.0% of im-

⁴ FPS is omitted as they are same as in Table 4.2.

provement both in AP and F1 scores on all experiments. In this setting, even the configuration of Experiment 7 could be used as a slightly error-prone yet fast enough detector in a real use-case, which outperforms the former studies. With the improved accuracy thanks to data distillation, Faster R-CNN with ResNet-50-FPN backbone using an input image size of 1200 pixels and region proposals up to 1000 enjoys an F1 score of 0.937 and AP score of 0.928 while offering 9.62 FPS.

Table 4.3: Accuracy scores after Data Distillation

Experiment	Meta-Architecture	Input Size	Region Proposals	AP	F1
1	MR-CNN ^a	1200	1000	0.9567	0.961
2	MR-CNN	1100	1000	0.914	0.938
3	MR-CNN	1200	50	0.829	0.842
4	MR-CNN	1100	50	0.748	0.769
5	FrR-CNN ^b	1200	1000	0.928	0.937
6	FrR-CNN	1100	1000	0.876	0.898
7	FrR-CNN	1200	50	0.775	0.81
8	FrR-CNN	1100	50	0.7	0.711

^a Mask R-CNN

^b Faster R-CNN

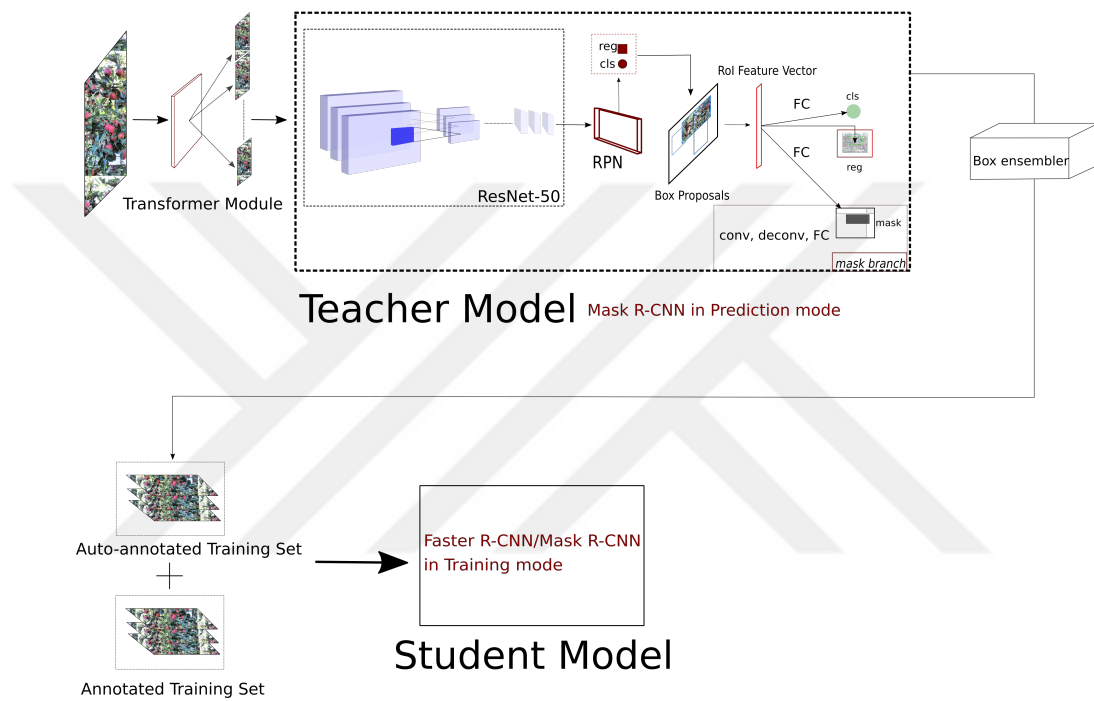


Figure 4.14: Combined view of Teacher and Student models in Data Distillation setting



CHAPTER 5

SOFTWARE ARCHITECTURE

The software stack consists of several ROS nodes dedicated to different tasks, such as planning and controlling the vehicles and fruit recognition. Each such node continuously communicates with a subset of other nodes to fulfill their goals. Additionally, there are a number of Gazebo plugins that set up sensor, actuator and communication interfaces required to conduct the simulation. These plugins standardize the sensory and actuary I/O both for PX4 autopilot and the [QuadManipulator](#) stack implemented in the scope of this thesis. The general layout of the software components of the overall system is drawn in Figure 5.1. In addition to that, Figure 5.2 sketches the detailed interactions between those components and how certain motion primitives alter the states of the aerial manipulator.

The detailed finite state machine in Figure 5.2 visualizes how each component interacts with the other. Also to note, it is visible in Figure 5.1 that each robot has its dedicated sublayer of software components to fulfill its tasks. In a nutshell, GCS sends a list of trees to each aerial manipulator. Then each robot sorts this list based on their distance and visits in a FIFO manner to harvest the apples until there are no more trees to visit. If there are still trees to visit, the robot proceeds to the closest 3D location on the orbital trajectory and starts to orbit for a finite amount of radians. Simultaneously the fruit detection module is enabled so as to find apples on the tree. If an apple is found, the orbit task gets preempted and the fruit picking task chain initiates. Consecutively, the [Arm Planner](#) module plans a trajectory for the 3DOF manipulator, which is then fed to the [Arm Executor](#). Thereafter the robot closes the grippers for a fixed amount¹. Again no feedback is used to determine whether the

¹ It must be noted here that a real application must use force feedback and accordingly adjust the distance between grippers to not harm the external structure of the fruit.

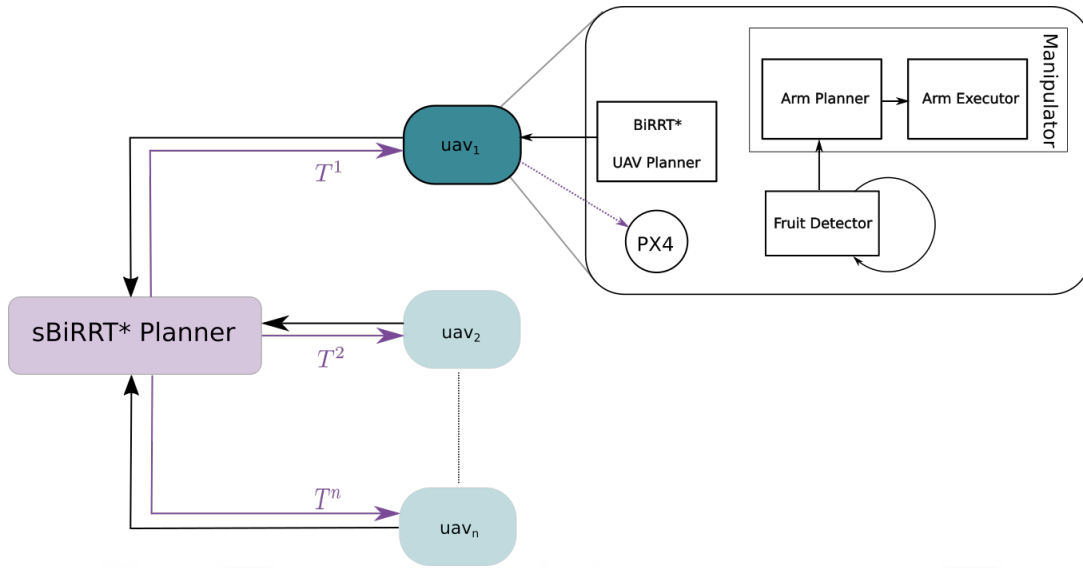


Figure 5.1: Overall System Architecture

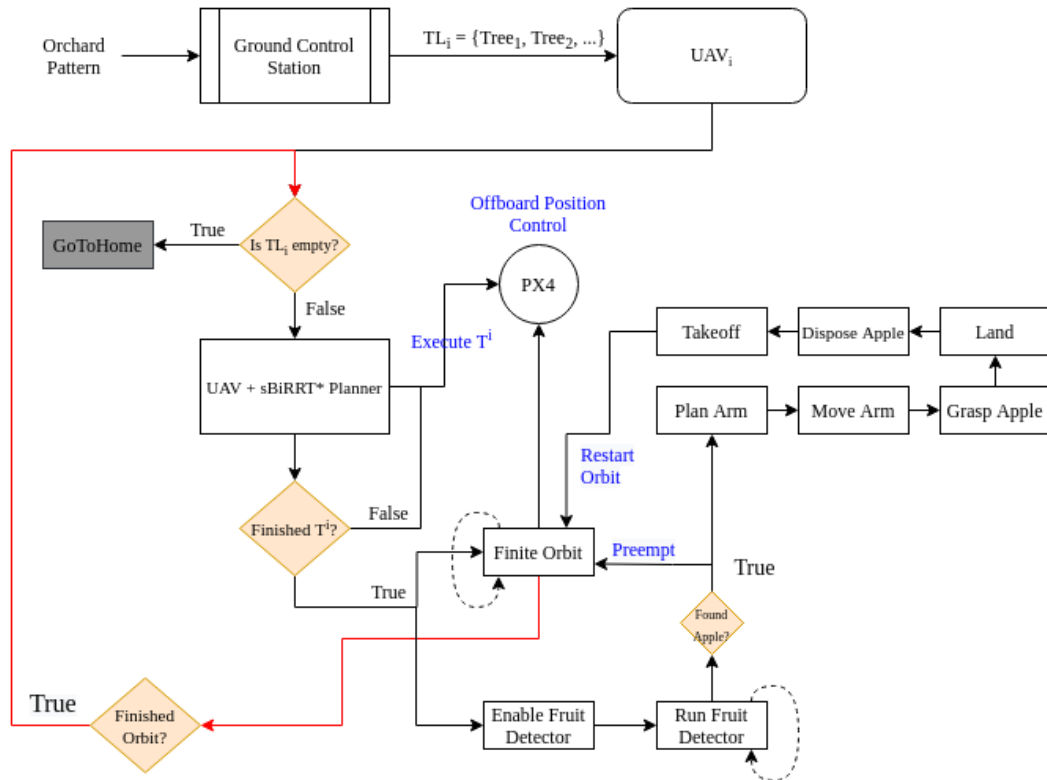


Figure 5.2: Overall System Finite State Machine

fruit is actually grasped and thus, the fruit is assumed to be picked with the closed grippers. Followingly the robot lands at the closest appropriate location and disposes the grasped apple. Finally, the preempted orbit task gets restarted and this loop continues until no apples are detected in a complete finite orbit.

5.1 Simulation

The simulation medium is Gazebo[13] software. Different `world` files -along with their `launch` files- are specified to test various aspects of the software such as orchard patrol, grasp & dispoose and fruit recognition. Finally there are `world` files dedicated to each scenario that are used as integration and regression test mediums for the ultimate use-case.

5.2 PX4 Parameters

EKF2_* parameters in Table 5.1 are essential for the aerial manipulator to work reasonably in the simulation environment. First two parameters assume an almost ideal GPS sensor model. Following two parameters set the process noise of gyrometer prediction to the lowest possible value. Similarly, magnetometer is assumed to be almost ideal. Finally, the innovation test count on GPS position is set as 1 to quickly detect excessive innovations in GPS position, which would lead to robot instability unless handled. All in all, these parameters hardly resemble any realistic construct; however, they are crucial for the PX4 autopilot to work on the aerial manipulator properly. Aligned with the assumption in Section 1.2, the estimation parameters are tuned for simulation to have reasonably sufficient control.

5.3 TF

The transformation tree is set up according to ROS REP105 [63] specification titled as *Coordinate Frames for Mobile Platforms*. It involves multiple robots, each of which does have its own instance of PX4 ROS node.

Table 5.1: PX4 Parameters

Name	Value	Description
NAV_RCL_ACT	1	Set RC loss failsafe mode to Hold
COM_OBL_ACT	1	Set offboard loss failsafe mode to Hold
NAV_DLL_ACT	1	Set data link loss failsafe mode to Hold
RTL_RETURN_ALT	10	Set Return mode altitude
COM_DISARM_PRFLT	-1	Time-out for auto disarm if not taking off disabled
NAV_ACC_RAD	2.0	Navigation acceptance radius in Mission mode
MPC_XY_CRUISE	1.5	Maximum horizontal velocity in Mission mode
EKF2_GPS_P_NOISE	0.01	Measurement noise for GPS position
EKF2_GPS_V_NOISE	0.01	Measurement noise for GPS velocity
EKF2_GYR_B_NOISE	0.01	Process noise for IMU rate gyro bias prediction
EKF2_GYR_NOISE	0.01	Rate gyro noise for covariance prediction
EKF2_MAG_NOISE	0.01	Measurement noise for magnetometer 3-axis fusion
EKF2_AID_MASK	5	Uses GPS data if available and inhibits IMU delta velocity bias estimation
EKF2_GPS_P_GATE	1	Gate size for GPS horizontal position fusion

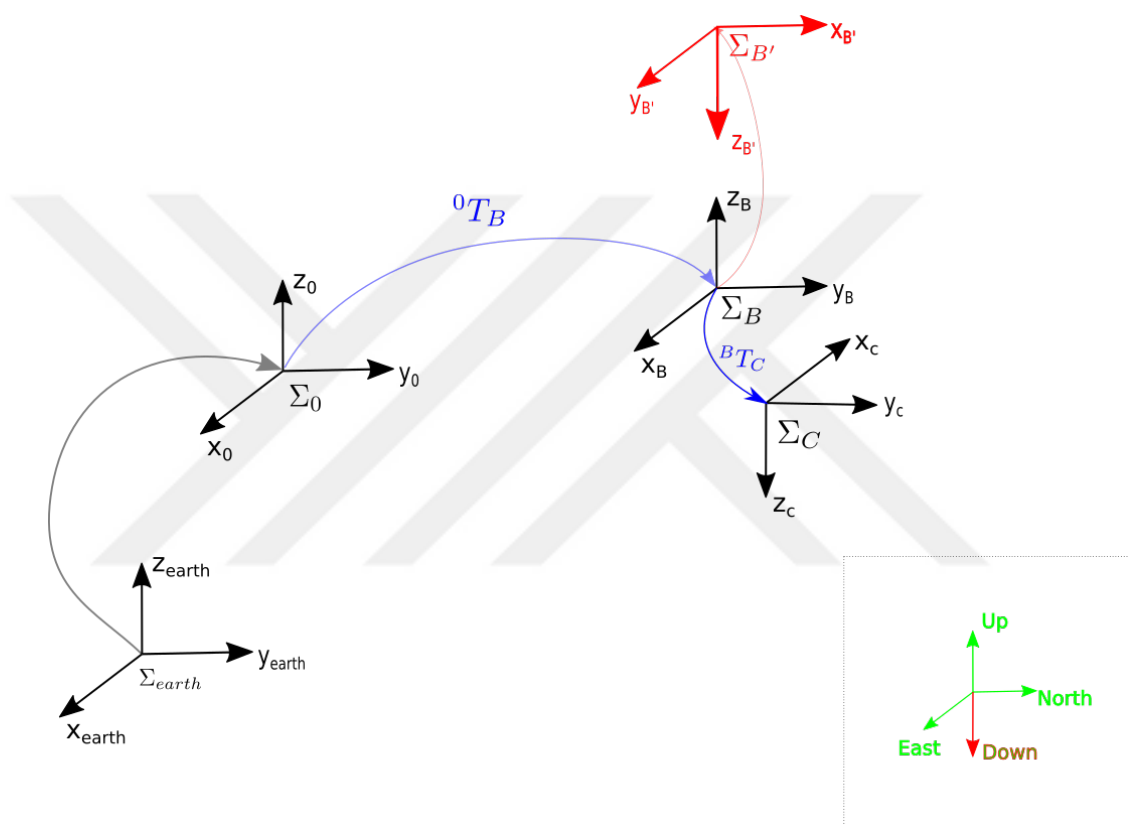


Figure 5.3: Transformation Tree with PX4 NED Frame

Figure 5.3 represents the transformation tree of a single robot. Σ_{earth} frame is consistent for all the robots and allows them to reason about each other's pose, essentially unifying the whole transform graph created by several robots into a single transform tree via becoming its root node. Σ_0 is the map frame of the particular robot. It gets started to be broadcasted in the bringup phase of the robot, where it is determined by the robot's initial pose with respect to Σ_{earth} , i.e. $^{earth}T_0$. Once retrieved, it is statically published and not subject to change. Naturally, it is different for each robot as it depends on the initial pose of the respective robot and thus prefixed with robot's namespace, e.g. `uav1`.

ROS uses ENU convention (East-North-Up $\rightarrow$ x-y-z) for Σ_{earth} , Σ_0 and Σ_B frames. On the other hand, PX4 employs NED convention (North-East-Down $\rightarrow$ x-y-z) on the body frame of the quadcopter — $\Sigma_{B'}$. This transformation needs to be applied to the data computed in the `QuadManipulator` stack that uses ROS convention on coordinate frames when sending to PX4 autopilot.

$${}^BT_{B'} = R_x(\pi) * R_z(\pi/2) \quad (5.1)$$

Ultimately, `odom` $\rightarrow$ `base_link` transformation needs to be briefly discussed. PX4 autopilot uses EKF2 to estimate ${}^0T_B$ by fusing multiple sensors already mentioned, e.g. GPS and IMU. It uses ${}^{odom}T_B$ sensor readings from the aforementioned odometry sources in its estimation process, which is *localization* in this context. Figure 5.4 illustrates this mechanism only for `uav1`, but it applies for all other UAVs. Lastly, ${}^{odom}T_B$ sensor readings are locally consistent, whereas ${}^0T_B$ is subject to discontinuities, known as *localization jumps*. Fortunately, this thesis tunes explicitly for almost ideal estimation and observation conditions, which allow us to ignore such real-world challenges in its scope. Nevertheless, it is still a good practice to conduct manipulation planning in timestamped locally consistent odometry frames. Considering the apple picking scenario, the estimated ${}^BT_{fruit}$ for a particular apple is susceptible to localization jumps, which would cause the robot to miss the apple and fail to pick. On the other hand, ${}^{odom}T_{fruit}$ does not change since the robot is physically still at the same pose, which would lead to the same (ignoring sensor noise) sensor reading from

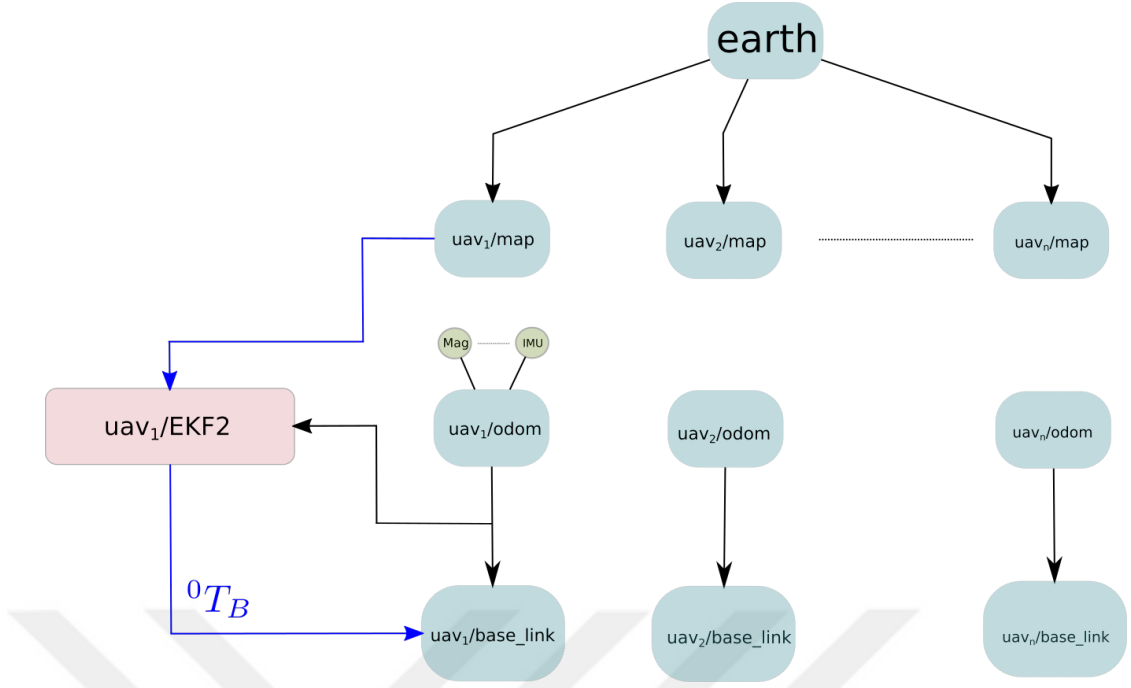


Figure 5.4: Unified Transformation Tree with Multiple Robots

the attached RGB camera.

5.4 QuadManipulator

The vehicle is governed by the `QuadManipulator` stack, which consists of an identically named C++ class for planning & execution of the overall vehicle -i.e. base and manipulator- and several Python scripts for fruit detection.

5.4.1 Motion Primitives

There are a number of motion primitives used in the fulfillment of higher-level tasks such as *patrol orchard* and *bring me an apple*. These are merely semantic descriptions of such types of tasks and this thesis does not provide any structured mechanism to create higher-level behaviors, e.g. via behavior trees [64]. Instead, it constructs simple finite state machines without any fallback mechanisms, meaningfully a failure in an earlier state would mean the failure of the entire scenario without the possibility of recovery actions.

5.4.1.1 Offboard Position Control

This is the most used motion primitive as it allows 4D (x, y, z, θ) control of the robot in Σ_0 . It is possible to define target poses in other frames as long as they are expressible in Σ_0 .

5.4.1.2 Takeoff

Takeoff is mainly used after the bringup phase of the robot; however, it can also be used after an emergency landing case. It makes the robot lift off to 1.5 meters above its initial location. GPS input is a prerequisite for this action; otherwise, PX4 denies taking off.

5.4.1.3 Landing

Lands the robot to the rigid base beneath it — ground, table, etc.

5.4.1.4 Go to Home

This is a composite motion primitive that uses *Offboard Position Control* and *Landing* to achieve its mission. It follows the routine in Algorithm 4. It first climbs up to 10 meters higher, then goes to the 2D origin of Σ_0 and consequently triggers a landing operation. It can be used at the end of scenarios or when some emergency case is encountered.

Algorithm 4 GoToHome

```
1: procedure GO_TO_HOME( ${}^0T_B$ )
2:    ${}^BT_{t_1} \leftarrow t_z(10)$ 
3:    ${}^0T_{t_1} \leftarrow {}^0T_B * {}^BT_{t_1}$ 
4:   move_base( ${}^0T_{t_1}$ )
5:    ${}^0T_{t_2} \leftarrow t_z({}^0T_{t_1}.z) * {}^0R_{t_1}$ 
6:   move_base( ${}^0T_{t_2}$ )
7:   land()
8:   return
9: end procedure
10: 

---


```

5.4.1.5 Orbit with Finite State Machine

Orbit is the most crucial motion primitive for tree inspection. The robot should keep a constant velocity profile and radius during tree inspection for the fruit detector to work reasonably. Also, the robot should not conduct orbit behavior infinitely, as is the case in [65]. Therefore a stateful custom flight task is introduced, whose execution status is reported by PX4 autopilot via MAVLink, which is translated to ROS through `mavros` node.

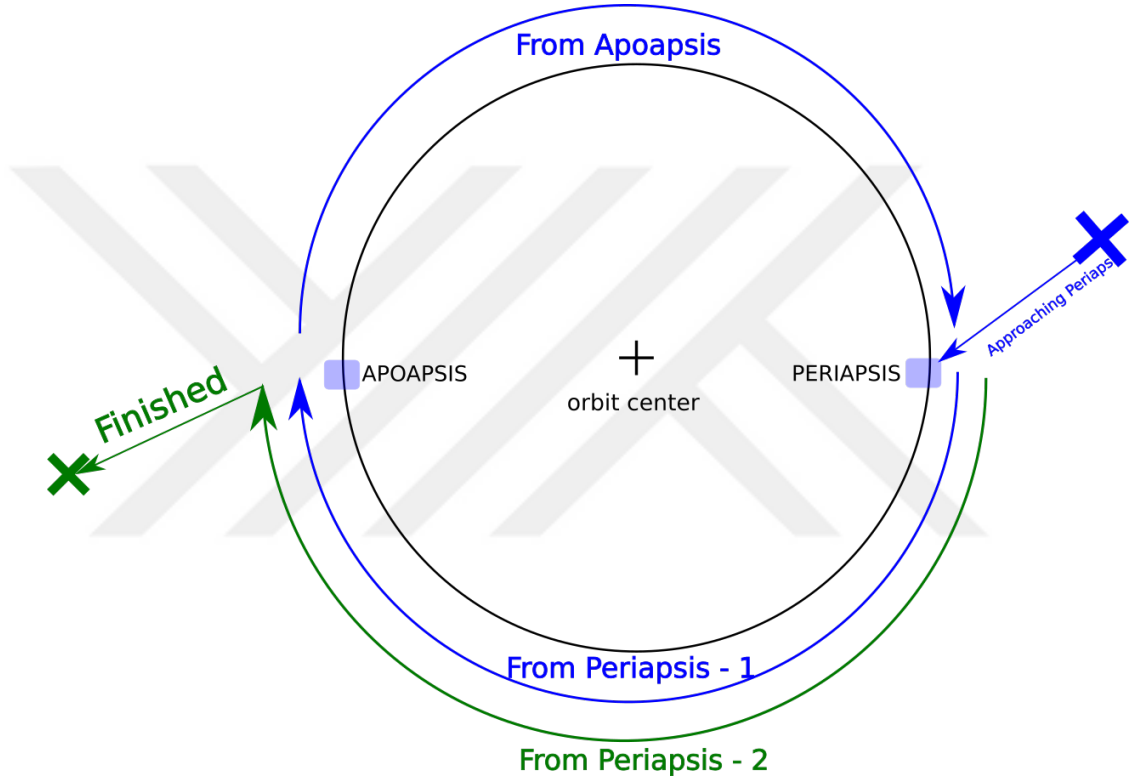


Figure 5.5: Orbit with Finite State Machine

Finite orbit behavior manifests the addition of states to originally stateless infinite orbit mode in PX4 autopilot. To that end, five new states are created to fulfill a circular trajectory of 3π along the orbit circle. Additionally, orbit mode requires the center pose in $\Sigma_{B'}$, orbit radius and orbiting velocity to be specified. Descriptions of states are:

- `APPROACHING_PERIAPSIS`: The robot will be in this state until it joins the

orbit circle from its initial pose.

- FROM_PERIAPSIS_1: The robot will transition into this state once it reaches PERIAPSIS. This state lasts for π radians along the orbit arc until APOAPSIS.
- FROM_APOAPSIS: The robot will transition into this state once it passes APOAPSIS. It lasts for π radians as well — until the robot reaches PERIAPSIS back.
- FROM_PERIAPSIS_2: The robot will transition into this state once it finishes one complete orbit cycle, effectively reaching PERIAPSIS again. It lasts for another π radians until it reaches APOAPSIS.
- FINISHED: The robot will start hovering once this state is reached, allowing the `QuadManipulator` stack to take control for further actions.

5.4.1.6 Plan Arm

${}^M T_{fruit}$ represents the detected fruit pose with respect to the attached manipulator base. Since it is possible to receive locally consistent and timestamped transforms of fruits as previously described, this task is straightforwardly achieved by `Arm Planner` using KDL Chain-IK Solver. At this level, it is imperative to have a fully stable robot - the rationale behind Chapter 2- since the `Arm Executor` component would blindly execute the planned trajectory from there on with `Move Arm` task.

5.4.1.7 Move Arm

The attached manipulator and grippers are position controlled; hence, they do not need to be controlled via a PID scheme. `JointTrajectoryController` from ROS literature is used, which computes a simple spline interpolation between the process and setpoint configurations of the arm.

5.4.1.8 Grasp & Dispose

Position control is generally impractical in real-world applications where the gripper setpoints are almost impossible to compute beforehand. However, we employ a Gazebo grasp plugin that handles these anomalies by rigidly attaching the external objects to one of the gripper links based on affecting forces. This, in return, allows us to “stop” moving grippers furthermore to a point they would squeeze out a possibly delicate fruit. The real-world solution is to use effort controllers.

5.4.2 Recognition

An object detection module runs as a separate ROS node with a simple on/off switch exposed via a ROS service `reco_cmd`. There are as many object detection nodes as the number of robots under the namespace of particular robots – e.g. `uav1/reco_cmd`. The object detection module is enabled after takeoff and disabled after landing so as to save the compute power. The object detection module devised in this study is not a general recognizer but merely a binary classifier on the bounded regions – apple or not. Therefore it cannot distinguish whether the robot is *close* to a tree by segmenting out the environment. For that reason, it is unconditionally enabled after takeoff. A more lightweight perception software could be employed as a gating module to extract “approaching to a tree” information in order to optimize battery usage. Nevertheless, this thesis focuses on a Proof of Concept (PoC), hence such practical concerns are left out of scope. Algorithm 5 is the modular apple detection algorithm, which is used both by baseline and machine learning inference modules.

5.4.2.1 OpenCV Baseline

In order to create a baseline, OpenCV[66] and Point Cloud Library (PCL)[67] are used in conjunction. A simulated RGB-D camera produces synchronized RGB and depth sensor readings at 20 Hz in the form of RGB images and point clouds respectively. A point cloud simply consists of a number of points expressed in Σ_C (optic center of the camera, see Figure 5.4) coordinate frame. After receiving the RGB im-

Algorithm 5 Apple Detection Algorithm

```
1: procedure DETECT_APPLES( $\lambda$ )
2:   Let  $TP = 0$  be number of true positives.
3:    $[C_{t_{p_1}}, C_{t_{p_2}}, \dots, C_{t_{p_n}}] \leftarrow \lambda()$ 
4:   for all  $C_{t_{p_i}}$  do
5:      $earth_{t_{p_i}} \leftarrow earth_{T_C} * C_{t_{p_i}}$ 
6:     if  $|earth_{t_{p_i}} - earth_{t_{f_j}}| < 0.02m \exists f_j$  then
7:        $TP \leftarrow TP + 1$ 
8:     end if
9:   end for
10:   $accuracy \leftarrow TP/total$ 
11:  return
12: end procedure
```

age and the corresponding point cloud, the image is converted to its HSV form and a simple red mask is created on this format. This mask effectively returns the coordinates of the pixels that are sufficiently red in the image frame. Only the points on the red-filtered image coordinates, of which further clipped under 2 meters of translational distance, are treated as detections after being transformed into Σ_{earth} . Finally, these detection results are published to `/orchard/detections` topic, on which the ground truth node subscribes. If the Euclidean distance between a detection and ground truth value of any fruit is shorter than 0.02 meters, it is considered a true positive. We compute the accuracy of the baseline detector based on this condition with different hyperparameters in Table 5.2.

Algorithm 6 OpenCV Baseline Apple Detector

```
1: procedure BASELINE_DETECTOR( $I_{RGB}, PC$ )
2:    $I_{HSV} \leftarrow I_{RGB}$ 
3:    $M_{red} \leftarrow clip(I_{HSV})$  ▷ Red-clip mask.
4:    $\lambda \leftarrow read(PC, M_{red}) \mid filter(2m)$ 
5:    $detect\_apples(\lambda)$ 
6:   return
7: end procedure
```

Table 5.2: Accuracy of OpenCV baseline detector with different orbit behaviors

Orbit Heights	Orbit Radius around Tree	Velocity in Orbit	Accuracy (Detected / Total) ^a
1.2m-2.5m	1.2m	0.4 m/s	57.8% (74.6/129)
1.2m-2.5m	1.2m	0.8 m/s	61.5% (79.4/129)
1.2m-2.5m	1.0m	0.4 m/s	56.2% (72.6/129)
1.2m-2.5m	1.0m	0.8 m/s	35.5% (45.8/129)

^a Average of 5 runs

All these experiments employ a 3x3 orchard consisting of entirely different trees. This comparison is helpful for the actual fruit recognition module such that how the velocity in orbit and orbit radius affects the true positive detection would facilitate the selection of correct combinations of parameters. It is clear from Table 5.2 that as the robot gets closer to trees and orbits more quickly, the accuracy reduces. However, the second case is interesting as the robot achieves an accuracy of 61.5% with 0.8 m/s velocity in orbit and 1.2m orbit radius, which is higher than the experiment with the same orbit radius and 0.4 m/s velocity in orbit. We can conclude that the relationship in here is non-linear from the perspective of velocity in orbit as soon as orbit radius exceeds some threshold value, which is probably between 1.0 and 1.2 meters.

5.4.2.2 Fast DeepApple

Object detection module from Chapter 4 is simply given as a parameter into `detect_apples` method in Algorithm 7 after converting RGB image into BGR format.

Algorithm 7 Fast DeepApple Detector

```

1: procedure FAST_DEEPPAPPLE_DETECTOR( $I_{RGB}$ )
2:   Let  $F$  represent the Mask R-CNN module developed in Chapter 4.
3:    $I_{BGR} \leftarrow I_{RGB}$ 
4:    $\lambda \leftarrow F(I_{BGR})$ 
5:   detect_apples( $\lambda$ )
6:   return
7: end procedure
8: 
```



CHAPTER 6

CONCLUSION

The ever-increasing adoption of quadcopters in a variety of use-cases renders them as perhaps the most ubiquitous robots in our everyday lives. Nevertheless, their true potential is hidden behind a cliff of a particular capability – manipulation. The fledgling field of aerial manipulation aspires to unlock this potential through solving challenging problems such as design, integration, control and motion planning of the aerial manipulators, which are complex systems that demand the simultaneous consideration of airframe and attached manipulator. Agricultural robotics and precision agriculture are such domains, which attracted excessive interest due to the deepening labor shortages, increasing labor costs and environmental concerns. The ever-growing human population, limited farming areas and ongoing global warming create an inevitable urge on the automation and digitization of agriculture, which must be rendered much more productive and environment-friendly at the same time. For instance, laser/steel equipped UGVs eliminate weeds precisely without any herbicides or pesticides[6]. In this thesis, we focused on the apple harvesting problem in order to create a PoC system that can be generalized to other fruits.

In Chapter 2, aerial manipulator dynamics are briefly narrated through Figure 2.2, which sketches the force/torque profiles affecting the center of the airframe when the attached manipulator is in motion. Since only hovering manipulation is in the scope of this thesis, the cases where also the airframe is navigating are not considered. After a small discussion on dynamics, integration of aerial manipulator via PX4 autopilot software through ROS and Gazebo is introduced. Since this thesis verifies the PoC only in simulation and approaches the problem solely from motion planning, fruit

recognition and multi-robot management perspectives; certain sensors, e.g. noise process of IMU, are tuned to end up with a sufficiently stable vehicle. While going in that direction, a solid assumption on the existence of an adequate control system is made for the real world.

Chapter 3 focuses on motion planning components. Motion planning is a classical AI field, with novel approaches brought through machine learning in the last decade. In this thesis, we started with a single robot motion planning problem through RRT algorithm family. Subsequently, we have continued with the introduction of a novel multi-robot variant sBiRRT* that employs sub-dimensional expansion, exactly as sRRT[31] does. This algorithm is optimal in the sense that it unifies the state-spaces of the respective robots only when there is a case of robot-robot collision. Once resolved, all planned policies of robots kept as they are until the next robot-robot collision, if any. The planned robot-robot and robot-object collision free paths are asymptotically optimal in the specification of the particular problem, which “labels” any intersecting paths as invalid, since there is no notion of time in it. Therefore the planned paths that are sub-optimal in the complete configuration state can be considered as optimal in the constrained problem. Nevertheless, truly optimal¹ paths can be computed with the proper inclusion of *time* into computation like in the Velocity-Tuning method[28].

Chapter 4 discourses the object detection problem in chronological order until the state-of-art. It introduces ResNet architecture and R-CNN machine learning model family, which revolutionized the mindset regarding the object detection problem through achieving the highest scores and winning contests. Mask R-CNN, which is the most recent and successful model that employs ResNet-50 and ResNet-101 backbones, is taken as the baseline for this study. Transfer learning is conducted on the pretrained COCO weights with a reannotated 800-image apple dataset. After receiving baseline accuracy scores, we opted for inference speedup techniques such as resizing the input image and limiting the number of proposals, which poses a tradeoff between speed and accuracy. Finally, the reduced accuracy scores due to speedup are regained with the Data distillation method, a semi-supervised machine learning technique that can

¹ well, asymptotically

utilize unannotated images via producing auto-annotations of them with a teacher model as to train a student model with the union set of auto-annotated and already annotated images.

The overall system architecture, the integration of modules and submodules, motion primitives and atomic tasks that effectively actuate the aerial manipulator, coordinate frame conversions to PX4 autopilot conventions, tuned PX4 parameters and bird-eye view of necessary transformations are studied in Chapter 5. In the end, we showcased a PoC system of systems that enable automated apple harvesting through aerial manipulation with a team of robots. Although apple harvesting (*or fruit harvesting, in general*) doesn't necessarily involve ecologically harmful matters, it is of utmost importance to optimize the agricultural costs in these type of already green use-cases so that other subjects -e.g. weed control- that might have possibly more expensive but green solutions can get room for adoption.



REFERENCES

- [1] G. Zhang, Y. He, B. Dai, F. Gu, J. Han, and G. Liu, "Robust control of an aerial manipulator based on a variable inertia parameters model." *IEEE Transactions on Industrial Electronics*, 2019.
- [2] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction with quadrotor teams." *Autonomous Robots*, 33(3), 323–336., 2012.
- [3] X. DING, P. GUO, K. XU, and Y. YU, "A review of aerial manipulation of small-scale rotorcraft unmanned robotic systems." *Chinese Journal of Aeronautics*., 2018.
- [4] M. Orsag, C. Korpela, P. Oh, and S. Bogdan, *Aerial Manipulation*, ser. Advances in Industrial Control. Springer International Publishing AG, 2018.
- [5] R. Tomorrow. Agrobot: Strawberry harvesters. [Online]. Available: <https://www.roboticstomorrow.com/story/2015/04/agrobot-strawberry-harvesters/5950>
- [6] R. Biz. Top 9 advanced agricultural robots for weed control. [Online]. Available: <https://roboticsbiz.com/top-9-advanced-agricultural-robots-for-weed-control>
- [7] S. Birrell, J. Hughes, C. J. Y., and F. Iida, "A field-tested robotic harvesting system for iceberg lettuce," *Journal of Field Robotics*, 2018.
- [8] [Online]. Available: <https://metricsproject.eu/agri-food>
- [9] C. E. Enyoh, A. W. Verla, W. Qingyue, D. K. Yadav, A. H. Chowdhury, B. O. Isiuku, T. Chowdhury, F. C. Ibe, E. N. Verla, and T. O. Maduka, "Indirect exposure to novel coronavirus (sars-cov-2): An overview of current knowledge," *European Commission Directorate-General for Health and Food safety, Food Hygiene Report*, 2020.

- [10] E. C. D.-G. for Health and F. safety. Covid-19 and food safety. [Online]. Available: https://ec.europa.eu/food/sites/food/files/safety/docs/biosafety_crisis_covid19_qandas_en.pdf
- [11] H. Utz and F. Stulp, “Sharing belief in teams of heterogeneous robots,” *Robot Soccer World Cup VIII*, 2004.
- [12] [Online]. Available: https://github.com/zhonghang1187151422/quad_manipulator
- [13] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.
- [14] P. Corke, *Robotics, Vision and Control Fundamental Algorithms in MATLAB 2nd Edition*, ser. Springer Tracts in Advanced Robotics. Springer International Publishing AG, 2017.
- [15] M. D. Ardema, *Newton-Euler Dynamics*. Springer Science+Business Media, Inc., 2005.
- [16] J. J. Craig, *Introduction to Robotics*. Addison Wesley, 1986.
- [17] L. Meier, D. Honegger, and M. Pollefeys, “Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.
- [18] [Online]. Available: <http://wiki.ros.org/mavros#mavros>
- [19] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, “Micro air vehicle link (mavlink) in a nutshell: A survey,” *IEEE Access*, vol. 7, pp. 87 658–87 680, 2019.
- [20] D. Brescianini, H. M., and D. R., “Nonlinear quadrocopter attitude control,” Institute for Dynamic Systems and Control (IDSC) ETH Zuurich, Tech. Rep., October 2013.

- [21] [Online]. Available: https://docs.px4.io/master/en/advanced_config/tuning_the_ecl_ekf.html
- [22] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Tech. Rep., 1998.
- [23] B. Chandler and M. A. Goodrich, “Online rrt* and online fmt*: Rapid replanning with dynamic cost,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6313–6318.
- [24] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.
- [25] B. Bilge, “Rrt-based kinodynamic motion planning for multiple camera industrial inspection,” 2009, unpublished thesis.
- [26] M. Jordan and A. Perez, “Optimal bidirectional rapidly-exploring random trees,” 2013.
- [27] K. Solovey, O. Salzman, and D. Halperin, “Finding a needle in an exponential haystack: Discrete rrt for exploration of implicit roadmaps in multi-robot motion planning,” *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 501–513, 2016.
- [28] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
- [29] G. Wagner, “Subdimensional expansion: A framework for computationally tractable multirobot path planning,” Ph.D. dissertation, The Robotics Institute Carnegie Mellon University, 2015, doctoral dissertation.
- [30] “Subdimensional expansion for multirobot path planning,” *Artificial Intelligence*, vol. 219, pp. 1–24, 2015.
- [31] G. Wagner, Minsu Kang, and H. Choset, “Probabilistic path planning for multiple robots with subdimensional expansion,” in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 2886–2892.

- [32] J. Elseberg, S. M. Rol, and S. A. Nüchter, “Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration,” *Journal of Software Engineering for Robotics*, 2012.
- [33] J. Kuffner and S. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.
- [34] [Online]. Available: <https://www.mathworks.com/help/matlab/ref/fminsearch.html>
- [35] H. Bruyninckx, “Orocos–open robot control software,” 2002.
- [36] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2010.
- [37] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: common objects in context,” *ECCV*, 2014.
- [38] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, “Object detection with deep learning: A review,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.
- [39] A. Khan, A. Sohail, U. Zahoora, and A. Saeed, “A survey of the recent architectures of deep convolutional neural networks,” *Artificial Intelligence Review*, vol. 53, 12 2020.
- [40] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *ICLR*, 2015.
- [41] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CVPR*, 2016.
- [42] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *CVPR*, 2014.

- [43] B. Alexe, T. Deselaers, and V. Ferrari, “Measuring the objectness of image windows,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2189–2202, 2012.
- [44] R. B. Girshick, “Fast R-CNN,” *ICCV*, 2015.
- [45] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” *NIPS*, 2015.
- [46] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 386–397, 2020.
- [47] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, “Detectron2,” <https://github.com/facebookresearch/detectron2>, 2019.
- [48] L. Fu, Y. Majeed, X. Zhang, M. Karkee, and Q. Zhang, “Faster r-cnn-based apple detection in dense-foliage fruiting-wall trees using rgb and depth features for robotic harvesting,” *Biosystems Engineering*, vol. 197, pp. 245–256, 2020.
- [49] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, “Speed/accuracy trade-offs for modern convolutional object detectors.” *CVPR*, 07 2017, pp. 3296–3297.
- [50] I. Radosavovic, P. Dollar, R. Girshick, G. Gkioxari, and K. He, “Data distillation: Towards omni-supervised learning.” *CVPR*, 06 2018, pp. 4119–4128.
- [51] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.
- [52] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” *NIPS*, 2014.
- [53] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial transformer networks,” *NIPS*, 2015.
- [54] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944.

- [55] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. Mccool, “Deepfruits: A fruit detection system using deep neural networks,” *Sensors*, vol. 16, p. 1222, 08 2016.
- [56] P. Chu, Z. Li, K. Lammers, R. Lu, and X. Liu, “Deepapple: Deep learning-based apple detection using a suppression mask R-CNN,” 2020. [Online]. Available: <https://arxiv.org/abs/2010.09870>
- [57] W. Dong, P. Roy, C. Peng, and V. Isler, “Ellipse r-cnn: Learning to infer elliptical object from clustering and occlusion,” *IEEE Transactions on Image Processing*, vol. 30, pp. 2193–2206, 2021.
- [58] F. Nogueira, “Bayesian Optimization: Open source constrained global optimization tool for Python,” 2014. [Online]. Available: <https://github.com/fmfn/BayesianOptimization>
- [59] T. G. Dietterich, “Ensemble methods in machine learning,” in *Proceedings of the First International Workshop on Multiple Classifier Systems*, ser. MCS ’00. Berlin, Heidelberg: Springer-Verlag, 2000, p. 1–15.
- [60] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 535–541.
- [61] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in *NIPS Deep Learning and Representation Learning Workshop*, 2015.
- [62] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” *International Conference on Learning Representations (ICLR) (Banff)*, 12 2013.
- [63] [Online]. Available: <https://www.ros.org/repos/rep-0105.html#id9>
- [64] M. Colledanchise and P. Ögren, “Behavior trees in robotics and AI: an introduction,” *CoRR*, vol. abs/1709.00084, 2017. [Online]. Available: <http://arxiv.org/abs/1709.00084>

- [65] [Online]. Available: https://docs.px4.io/master/en/flight_modes/orbit.html
- [66] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
- [67] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

