# View Point Planning for Inspecting Static and Dynamic Scenes with Multi-Robot Teams

Ashish Kumar Budhiraja

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

July 24, 2017
Blacksburg, Virginia

Keywords: Multi-Robot Coordination, Traveling Salesman Problem, Target Tracking

# View Point Planning for Inspecting Static and Dynamic Scenes with Multi-Robot Teams

Ashish Kumar Budhiraja

## ABSTRACT

We study the problem of viewpoint planning in static and dynamic scenes using multi-robot teams. This work is motivated by two applications: bridge inspection and environmental monitoring using Unmanned Aerial Vehicles. For static scenes, we are given a set of target points in a polygonal environment that must be monitored using robots with cameras. The goal is to compute a tour for all the robots such that every target is visible from at least one tour. We solve this problem optimally by reducing it to Generalized Travelling Salesman Problem. For dynamic scenes, we study the multi-robot assignment problem for multi-target tracking. The problem can be viewed as the mixed packing and covering problem. We optimally solve the problem using Mixed Quadratic Integer Linear Program to maximize the total number of targets covered. In addition to theoretical contribution, we also present our hardware system design and findings from field experiments.

This material is based upon work supported in part by the National Science Foundation under Grant No. 1566247 and NIFA grant 2015-67021-23857.

# View Point Planning for Inspecting Static and Dynamic Scenes with Multi-Robot Teams

Ashish Kumar Budhiraja

## GENERAL AUDIENCE ABSTRACT

We study the problem of viewpoint planning in static and dynamic scenes using multi-robot teams. This work is motivated by two applications: bridge inspection and environmental monitoring using Unmanned Aerial Vehicles. For static scenes, we are given a set of target points in a static 2D or 3D environment such as a bridge. Target points are key locations that we are interested to monitor using cameras on the robots. The goal is to compute a tour for all the robots such that every target location is visible from at least one robot's tour. We want to minimize the sum of lengths of all the robot's tours combined. We find the best possible solution for this problem. For dynamic scenes, we study the multi-robot trajectory assignment problem for multi-target tracking. Here, the target points may be moving, e.g., expanding plumes in an oil spill. The goal in this is to maximize the total number of targets covered at each time step. We provide the best possible solution in this case. In addition to theoretical contribution, we also present our hardware system design and findings from field experiments.

This material is based upon work supported in part by the National Science Foundation under Grant No. 1566247 and NIFA grant 2015-67021-23857.

## Dedication

To my family.

# Acknowledgments

I would first like to thank my advisor, Dr. Pratap Tokekar, who has supported me throughout with his patience and knowledge. I thank him very much for assisting me every time with all the challenges and difficulties. I really appreciate all his contributions, by way of his time and ideas, through which I have learned so much in the last 2 years. I attribute much of the work I completed towards my Master's degree to his encouragement and effort. One simply could not wish for a better or friendlier supervisor.

In my daily work at Robotics Algorithms and Autonomous Systems lab, I have been blessed with a friendly and cheerful group of fellow students. I would like to thank Kevin Yu, Nahush Gondhalekar, Aravind Premkumar, Yoonchang Sung, Varun Suryan, Zhongshun Zhang and Lifeng Zhou for all the support and good times. Field experiments and research work has always been fun because of their help.

A sincere thanks to my friend Kevin, who has been just a call away in case of need. In many ways I have learnt much from, and because of him!

Finally, I thank my parents and my wife for supporting me through all my ups and downs. Their support and encouragement has always been my driving force, and without it, this would not have been possible.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

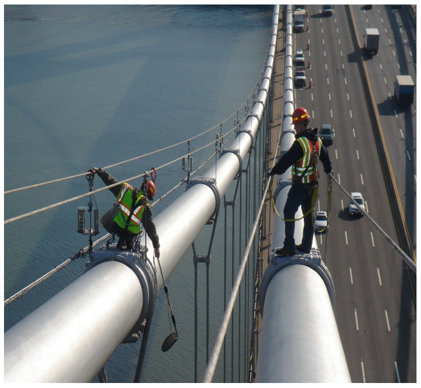
In recent years, interest in Unmanned Aerial Vehicles (UAVs) has grown rapidly thanks to the availability of lighter and compact sensors, lower cost of components and advancement in battery technology. There are many off-the-shelf products available [19]. These products provide some autonomous functionality e.g., "Position Hold" and GPS based waypoint navigation. While these functionalities are useful in some applications, in order to make full use of capabilities of UAV's we need to advance the autonomous decision-making capabilities. In this thesis, we present view point planning algorithms that do just that.

Our work is motivated by applications such as inspection of civil infrastructure and environmental monitoring where UAVs are used as mobile cameras. For effective inspection, we need algorithms that find paths that optimize the views obtained by the UAVs. For tracking applications, we need dynamic planning, since we cannot use existing solutions of clicking points on the map to plan paths. This task is even more complicated when multiple UAVs need to coordinate. The goal of this thesis is to address challenges associated with static and dynamic view point planning for multi-robot teams.

## 1.1 Motivating Applications

Our main motivation for static view point planning is the application of UAVs to inspect bridges and other civil infrastructure. This task is currently performed manually using harnesses from bridge (see Figure 1.1(a)) or using cranes (see Figure 1.1(b)). Both methods have significant risk and can be time-consuming (see Figure 1.1(c)). Data in the form of images and videos can be used to assess the structural integrity of the bridge. UAVs become a prime candidate to address this problem as they can collect a large amount of data in a relatively short amount of time. They require a small crew and are relatively inexpensive. This could potentially save insurance expenses and infrastructure costs such as cranes [84].

Bridges typically have critical target points that need to be inspected, such points can be identified by experts. These points can be inside or outside the bridge structure. In both cases there are issues related to stability and control that are being addressed in the literature [15] [17] [49]. Given a navigation stack for UAVs similar to what is available for ground robots in Robot Operating System(ROS) [82], the next stage of monitoring target points for inspection in a bridge would be planning optimal tours for multi-robot teams. In Chapter 2, we discuss how we address this problem. Specifically, we provide an algorithm that gives an optimal tour for multiple UAVs so that every input point of interest is seen by at least one tour.


(a) Manual inspection [10].

(b) Crane inspection [11].

(c) Routine inspection gone wrong [55].

Figure 1.1: Bridges are typically inspected manually using harnesses or cranes. Both methods carry significant risk for human life and property.

(a) Skimmers come in various designs but all basically work by removing the oil layer from the surface of the water (U.S. Coast Guard) [54].

(b) Burning oil "in place" (in situ) on the water's surface requires gathering a layer of oil thick enough to sustain the burn (NOAA) [54].

Figure 1.2: Two major methods of cleaning an oil spill at sea. Effectiveness of these methods depends on timely action, terrain and weather conditions [54].

For dynamic viewpoint planning, our main motivation is to use UAVs for environmental monitoring to effectively detect and track hazardous agents in aquatic environments. Sending humans alone to detect hazardous threats can be dangerous as well as an inefficient use of limited resources [53]. The ability to have a fast response team of emergency responders and robots that can detect, track and map expanding plumes of hazardous agents and respond accordingly is needed. As an example, in the case of oil spills the method to contain it changes from containment and skimming (See Figure 1.2(b)) to using sorbents and in situ burning in a matter of hours [29] (See Figure 1.2(b)). Hazardous agents like oil can be detected rapidly using UAVs. We need algorithms to define the interaction between aerial robots so that they can act as early warning systems for hazardous agents such as chemicals and radioactive particles.

Teams of UAVs flying over these aquatic environments can provide a wider picture of expanding plumes in water. Tracking these spatiotemporal plumes requires UAVs to be equipped with autonomous detection and tracking capabilities. Our aim is to use multiple robots so as to effectively track collection of targets in a moving environment. Target tracking can be further divided into two categories: 1) actively controlling the sensor position to improve estimation, and 2) estimating the position of a target using noisy sensor measurements. The first problem is addressed in Chapter 3 and the second problem is addressed in Chapter 4.

## 1.2 Contributions of this Thesis

Figure 1.3: Using a UAV to inspect the smart bridge at Virginia Tech [69].

Figure 1.4: UAV using a camera for environmental monitoring at Claytor Lake.

There are three main contributions of this thesis.

- We study the problem of computing tours for multiple robots in a polygonal environment, such that, every target is visible from at least one tour. We present a practical solution that finds an optimal solution in possibly exponential time. This work is presented in Chapter 2. A paper describing this work is currently under revision, after one round of reviews, at the IEEE Transaction on Robotics [76].
- We provide an optimal baseline to the multi-robot assignment problem for multi-target tracking. This problem can be viewed as mixed packing and covering problem. We optimally solve the problem using Mixed Quadratic Integer Linear Program [80] to maximize the total number of targets covered. This work is presented in Chapter 3 and is planned to be submitted to a conference [72].
- We present hardware and software design for a quadrotor platform based on off-the-shelf components that are suitable for the two applications mentioned above. We also present results from preliminary field experiments. The design and preliminary experiments are presented in Chapter 4. Figure 1.3 shows our inspection experiments at Virginia Tech Transportation Institute's (VTTI) smart road project [69]. Figure 1.4 shows our UAV with a down-facing camera flying over Claytor Lake.
- We finally conclude with a discussion of future work in Chapter 5.

# Chapter 2

# View Point Planning For Static Scenes

## 2.1 Introduction

The problem addressed in this chapter is motivated by view point planning for bridge inspection, as described in Chapter 1. The algorithm presented solves for an optimal tour for multiple UAVs to see all target locations of interest. These target locations can be specific load points that are crucial to the safety of the bridge. The targets can be decided offline by expert civil engineers using existing maps of the bridges or by other methods. The preliminary experiments for bridge inspection are described in Chapter 4.

The above mentioned problem can be formalized as that of planning paths for a team of robots tasked with visually monitoring complex environments. Visibility-based monitoring problems commonly occur in many applications such as surveillance, infrastructure inspection [61], and environmental monitoring [70]. These problems have received significant interest recently [42, 71, 89], thanks in part, to the technological advances that have made it easy to rapidly deploy teams of robots capable of performing such tasks. For example, Michael et al. [50] demonstrated the feasibility of carrying out persistent monitoring tasks with a team of Unmanned Aerial Vehicles (UAVs) with onboard cameras.

A richer version of the problem where the points to be visited by the robots are not given instead must be computed based on visibility-based sensing is studied by Tokekar et. al. [78]. In our problem, we are given a set of target points in a polygonal environment. Each robot carries a camera and can see any target as long as the straight line joining them is not obstructed by the boundary of the environment. Our goal is to compute paths for $m$ robots, so as to ensure that each target is seen from at least one point. Figure 2.1 shows an example scenario for $m = 2$ robots, $x = 3$ targets, $v = 4$ viewpoints and $b = 2$ depot locations.

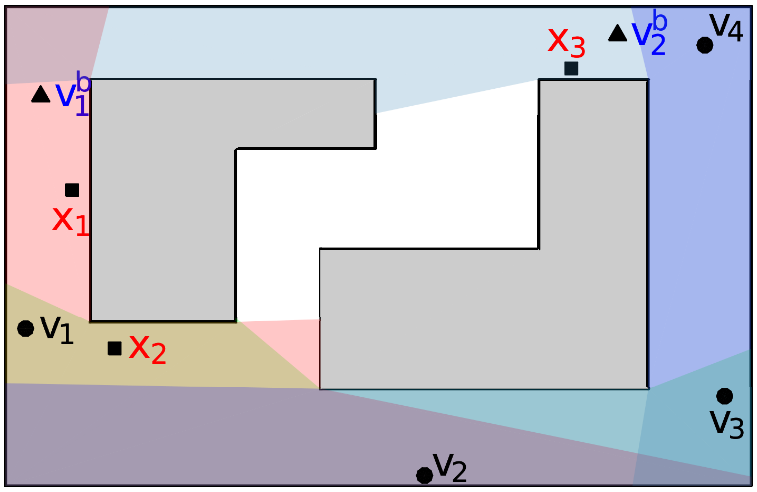


Figure 2.1: Problem formulation. We are given a set of target points $(x_i)$ in a polygonal environment. Also, we are given discrete viewpoints $(v_j)$ such that each target is seen from at least one viewpoint. $v_m^b$ is robot depot location. Our goal is to find $m = 2$ tours, one for each robot. The objective is to minimize the total length of tours traveled by robots.

## 2.2 Related Work

Persistent monitoring problems are typically studied when the points of interest are given as input. The points may have associated weights representing their importance. A common objective is to find the order of visiting the points that minimize the weighted latency. Alamdari et al. [3] showed that this problem is NP-hard and presented two log factor approximation algorithms. In many settings, the path to be followed by the robots is given as input as well, and the speed of the robot must be optimized to minimize the maximum weighted latency. Cassandras et al. [14] presented an optimal control approach to determine the speed profiles for multiple robots when their motion is constrained to a given curve. Yu et al. [88] presented an optimal solution for computing speed profiles for a single robot moving along a closed curve to sense the maximum number of stochastically arriving events on a curve. Pasqualetti et al. [62] presented distributed control laws for coordination between multiple robots patrolling on a metric graph.

Our problem is a generalization of the Art Gallery Problem (AGP) [60] and the Watchman Route Problem (WRP) [18]. The objective in AGP is to find the smallest set of “guard” locations, such that every point in an input polygon is seen from at least one guard. AGP is NP-hard for most types of input polygons [60], and very few approximation algorithms exist even for special cases. The objective in WRP is to find a tour of minimum length for a single robot (i.e., watchman) so as to see every point in an input polygon. There is an optimal algorithm for solving WRP in polygons without any holes [13] and a $\mathcal{O}(log^2 n)$ approximation algorithm for $n$-sided polygons with holes [51]. Carlsson et al [13] introduced $m$–WRP where the goal is to find $m$ tours such that each point in the environment is seen from at least one tour. The objective is to minimize the total length of $m$ tours. They showed that the problem is NP-hard (in fact, no approximation guarantee is possible).

Wang et al. [85] first introduced this objective function for WRP for the case of a single robot and termed it the Generalized WRP (GWRP). They showed that GWRP is NP-hard and presented a $\mathcal{O}(\text{polylog}n)$ approximation for the restricted case when each viewpoint is required to see a complete polygon edge.

Tokekar et. al. [78] introduced the $m$ robot version of GWRP. This problem, in general, is NP-hard since it generalizes the NP-hard problems of GWRP and $m$-WRP. Hence, they consider special instances of the problem and present a number of positive results. In particular, they characterize the conditions under which the problem has an optimal algorithm, in Section III of their paper and a present a constant-factor approximation algorithm for a special class of environments, in Section IV of their paper.

As an extension to the work done by Tokekar et. al. [78], we study the case when the measurement time for a sensor is zero (Chapter 2.3). We present a practical solution that finds paths for $m$ robots. Our solution can be applied for a broad class of environments (*e.g.*, 2.5D, 3D) and can incorporate practical sensing constraints (*e.g.*, limited sensing range, and field-of-view). The added generality comes at the expense of running time. Instead of a polynomial time solution, our algorithm may take possibly exponential time. We show how to use existing, sophisticated Traveling Salesperson Problem solvers to produce solutions in reasonable amounts of time (for typical instances).

## 2.3 Problem Description

In this section, we address the general case for the multi-robot watchmen route problem. We remove the restriction of street polygons and requiring a chain-visible curve as the input required by Tokekar et. al. [78]. However, the added generality comes at the expense of some relaxations. We assume that a finite set of candidate measurement locations, $V$, is given as input. The goal is to find tours for each robot visiting a subset of $V$ such that they collectively see all the targets. Note that there is no further assumption (*e.g.*, chain-visibility) on $V$. Consequently, $V$ can be computed by simply discretizing the environment either uniformly or using the strategy in Deshpande et al. [20]. Instead of minimizing the maximum times of the tour, we resort to minimizing the sum of the length of all tours (*i.e.*, $t_m = 0$).

Our contribution is to show how to solve this general version of the multi-robot watchman route problem by reducing it to a TSP instance. The resulting TSP instance is not necessarily metric, and consequently existing polynomial time approximation algorithms cannot be directly applied. Instead, we directly find the optimal TSP solution leveraging sophisticated TSP solvers (*e.g.*, Concorde [4]). This is motivated by recent work by Mathew et al. [46] who used a similar approach for solving a multi-robot rendezvous problem. We demonstrate that this algorithm finds the optimal solution faster than directly solving the original problem using an Integer Linear Programming solver.

In the following, we describe our reduction and prove its correctness. We also present empirical results from simulations for a 2D case. However, the following algorithm also works for 3D problems and can incorporate additional sensing range and motion constraints.

## 2.4 Reduction to TSP

Let $P$ denote the environment in which the targets points $X$ are located. We are given a set of candidate viewpoints $V$ within $P$. We denote the $j^{th}$ viewpoint by $v_j \in V$ where $j = \{1, ..., n\}$. For each target $x_i \in X$, we create cluster of viewpoints, $C_i = \{v_j \mid v_i \subseteq V \text{ can see } x_i\}$. Without loss of generality, we assume that $V$ is such that each target $x_i$ is seen from at least one viewpoint (*i.e.*, $|C_i| \geq 1$ for all $i$). One way of generating a valid set $V$ is by sampling or discretizing the visibility polygons for all $x_i$. For the special case, when $X$ is the set of all points in a 2D polygon, we can generate $V$ by imposing a grid inside $P$ using the strategy from [20].

If a robot visits any viewpoint in cluster $C_i$, then it ensures that target $x_i$ is seen by the corresponding robot. Therefore, the goal is to find a set of tours, one per robot, such that at least one viewpoint in each $C_i$ is visited. Note that the clusters need not be disjoint. This problem is equivalent to the multi-robot Generalized Traveling Salesman Problem (GTSP) [44].

The input to GTSP is a set of clusters, each containing one or more nodes from a connected graph. The goal in single robot GTSP is to find the minimum length tour that visits at least one node in each cluster.[1] GTSP is NP-hard [44] since it generalizes TSP.

Noon and Bean [56] and Lien and Ma [44] presented two polynomial time reductions of GTSP into a TSP instance such that the optimal TSP tour yields the optimal GTSP solution. The resulting TSP instance is not necessarily metric and consequently, the standard approximation algorithms (*e.g.*, [6]) cannot be applied. We can find the optimal solution of the TSP instance directly using potentially exponential time algorithms. A number of sophisticated implementations have been developed for solving large TSP instances to optimality [4]. In particular, we use the *Concorde* solver which yields the best-known solutions to large TSP instances [5]. In Chapter 2.5, we compare this approach with a generic Integer Linear Programming solver.

The reductions from GTSP to TSP proposed by Noon and Bean [56] and Lien and Ma [44] are for finding a single robot GTSP tour. In our case, we are interested in finding $m$ tours – one for each of the $m$ robots. Mathew et al. [46] presented a multi-robot extension for the Noon-Bean transformation. This transformation is applicable for the case when the clusters in the GTSP instance are disjoint (*i.e.*, $|C_i \cap C_j| = 0$, for all $i \neq j$). With slight modification, we can apply the

[1]There are versions of GTSP with additional restrictions of visiting exactly one node in each cluster or visiting each cluster exactly once. We consider the less restrictive version where the robot is allowed to visit a cluster multiple times while still ensuring a specific node is visited no more than once.

transformation to the case of possibly overlapping clusters as given below.

We assume that the path for the $i^{th}$ robot must start at a specific vertex, $v_d^i$, and end at a specific vertex $v_f^i$. We model three scenarios:

1. SAMEDEPOT: All robots start and finish their tours at the same location. That is, $v_d^i = v_d^j = v_f^i = v_f^j$ for all $i$ and $j$.
2. SAMEFINISHDEPOT: All robots finish their tours at the same location but may have unique starting locations. That is, $v_f^i = v_f^j$ for all $i$ and $j$.
3. INTERCHANGEABLEDEPOTS: There are $m$ fixed depots and initially, there is one robot at each depot. The robots can end their paths at any of the $m$ depots with the restriction that each depot must have one robot at the end.

For all the scenarios, the algorithm given below finds $m$ tours such that the sum of the lengths of all tours is minimized and each target is seen.

The reduction consists of five main steps. First, represent the given instance as a graph. Second, form a metric completion of the graph and remove viewpoints that cannot see any target. Third, convert the overlapping clusters to non-overlapping ones. Fourth, use a modified Noon-Bean reduction [56] to convert the GTSP instance into a TSP instance. Fifth, add start and finish depots as nodes in the graph. The detailed description of each step is given below.

1. (Figures 2.2(a)–2.2(b)) Represent $P$ as a graph $G^g = (V, E, \hat{C})$. The vertices are the viewpoints $V \in P$. Add an edge $(v_i, v_j) \in E$ where $v_i, v_j \in V$ and $v_i$ is visible from $v_j$. The cost of $(v_i, v_j)$ is the Euclidean distance between $v_i$ and $v_j$. Define a set of clusters $\hat{C} = \{C_i, \ldots, C_k\}$ where $C_i = \{v_j \mid v_j \in V \text{ is visible from target } x_i\}$.
2. (Figures 2.2(b) to 2.2(c)) Complete the graph $G^g$ in Step-1 to give $G^c = (V^c, E^c, \hat{C})$. Cost of an edge $(v_i, v_j)$ in $E^c$ is equal to the cost of shortest path between $v_i$ and $v_j$ in $G^g$. Then, remove isolated nodes $V^r$ that are not present in any cluster in $\hat{C}$. This in turn removes edges $E^r = \{(v_i, v_j) \in E^c \mid \forall v_i \in V^r\}$. Above operations give us $G^{cr}$=$(V^{cr}, E^{cr}, \hat{C})$ where $V^{cr} = V \setminus V^r$ and $E^{cr} = E^c \setminus E^r$. [2]
3. Carry out the first step in Noon-Bean transformation [56], the I-N transformation, as follows: $G^{cr}$=$(V^{cr}, E^{cr}, \hat{C})$ is converted to a graph $G^{in} = (V^{in}, E^{in}, \hat{C}^{in})$ that has non-intersecting set of clusters. We go through following steps:
   - Create set of nodes $V^{in} = \{v_{i,j} \mid \forall v_i \in V^{cr} \text{ and } \forall C_j \ni v_i, C_j \in \hat{C}\}$.
   - Create set of clusters $\hat{C}^{in}$=$\{C_j^{in}\}$ where $C_j^{in} = \{v_{i,j} \mid \forall i \in \{1, \ldots, n\}, v_{i,j} \in V^{in}\}$.

[2]The completion may result in some nodes to be visited multiple times in the final tour.

- For all nodes of the form $\{v_{i,j}, v_{i,k}\} \in V^{in}$ where $j \neq k$, create an 'intranode-intercluster' edge $(v_{i,j}, v_{i,k}) \in E^{in}$ and assign a zero cost to this edge.
- For all nodes of the form $\{v_{i,p}, v_{j,q}\} \in V^{in}$ create an edge $\{(v_{i,p}, v_{j,q}) \in E^{in}$ where $p \neq q$ and $i \neq j\}$, and assign the cost of this edge equal to the cost of corresponding edge $(v_i, v_j) \in E^{cr}$. Refer to Figures 2.2(c)–2.2(d).
- Choose $\alpha$ greater than the cost of any tour in $P$ that visits all targets $x_i$. Add this penalty $\alpha$ to all edges in $E^{in}$ except zero cost edges.[3] The exact expression for $\alpha$ will be defined in Step 5.

4. Complete the Noon-Bean reduction by adding intracluster edges, tail-shifting, and imposing another penalty. Convert $G^{in} = (V^{in}, E^{in}, \hat{C}^{in})$ to a new graph $G^t = (V^t, E^t, C^t)$ as follows:

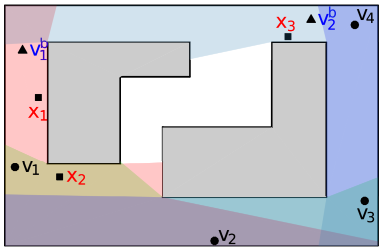
   - Copy the vertices, edges and clusters: $V^t = V^{in}$, $E^t = E^{in}$, and $\hat{C}^t = \hat{C}^{in}$. Create edges to connect all nodes in cluster $C_j^{in} \in \hat{C}^{in}$ by an intracluster cycle in any order. That is, $\{v_{i_1,j} \to v_{i_2,j} \to \ldots \to v_{i_p,j} \to v_{i_1,j} \mid \{v_{i_1,j}, \ldots, v_{i_p,j}\} \in \hat{C}_j^{in}, \forall j \in \{1, \ldots, k\}\}$. Here $v_{i_1,j} \to v_{i_2,j}$ represents an edge $(v_{i_1,j}, v_{i_2,j})$. To mark these edges we assign a cost of $-1$ to them. Add these edges to $E^t$.
   - For all intercluster edges $E^{tl} \in E^t$ where $E^{tl} = \{(v_{ip}, v_{jq}) \mid p \neq q, v_{ip} \in C_p^t, v_{jq} \in C_q^t, C_p^t, C_q^t \in \hat{C}^t\}$, we move the tail of all edges $(v_{ip}, v_{jq}) \in E^{tl}$ to the previous node $v_{i\text{-}1p}$ in the intracluster cycle defined above. That is, $(v_{ip}, v_{jq})$ changes to $(v_{i\text{-}1p}, v_{jq})$.
   - We choose a cost $\beta$ to be greater than the cost of any tour in the environment $P$ that visits all targets $x_i$ with a $\alpha$ penalty added to each of the edges in the tour. Add this penalty $\beta$ to all edges in $E^t$ except those marked with $-1$ costs. Replace the cost of all the edges with cost $-1$ to a cost of zero.

5. Add robot depots to the graph. Create a new graph $G^b = \{V^b, E^b\}$. For all the depots, add $v_i^d$ and $v_i^f$ to $V^b$. For the InterchangeableDepots we create two copies of each depot. For SameDepot we create $2m$ copies of the depot, and for SameFinishDepot we create $m$ copies of the finish depot. Create zero cost edges $E^b$ between all pairs of depot nodes. We get the final directed TSP graph $G^f = (V^f, E^f)$. Here $V^f = V^b \cup V^t$ and $E^f = E^b \cup E^t$. We also add the following edges to $E^f$:

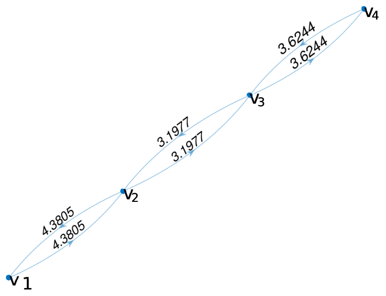
   - Create depot outgoing edges $E^o = \{(v_i^d, v_{jp}) \mid \forall v_i^d \in V^b, \forall v_{jp} \in V^t\}$. Assign a cost $\gamma_i$ to all edges $(v_i^d, v_{jp})$ equal to the cost of the shortest path connecting $v_i^d$ to $v_{jp}$ in $P$ restricted to $V$. Also, add a penalty $\alpha + \beta$ to these edges. Add $E^o$ to $E^f$.

[3]We use the cost of an arbitrary tour in the original GTSP for this step instead of using sum of the cost of all the edges as used in P2 of Noon-Bean Method [56]. This is used to prevent numerical issues with large penalties in Concorde.

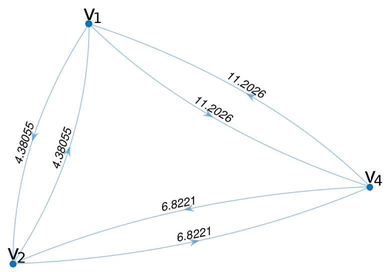
- Create tail shifted depot incoming edges $E^i = \{(v_{jp}, v_i^f \mid \forall v_{jp} \in V^t, \forall v_i^f \in V^b\}$. Also, add the cost $\gamma_i$ to $(v_{jp}, v_i^f)$ as above but without penalty. Then we move tail of all the edges $(v_{jp}, v_i^f) \in E^i$ to the previous node $v_{j\text{-}1p}$ in the intracluster cycle defined in Step 4 above. That is, $(v_{jp}, v_i^f)$ changes to $(v_{j\text{-}1p}, v_i^f)$. Add $E^i$ to $E^f$.
- Define $\alpha$=2(cost of MST in $G^{cr}$)+2$m$|cost in $E^i|_{max}$. Define $\beta$=2$\alpha$(1+(#edges in MST in $G^{cr}$))+2$m(1+\alpha)$.



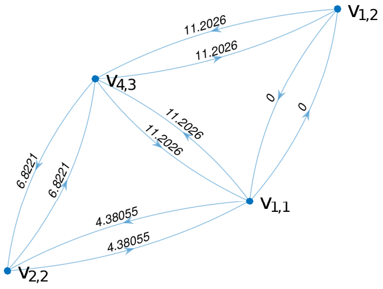
(a) Input Instance



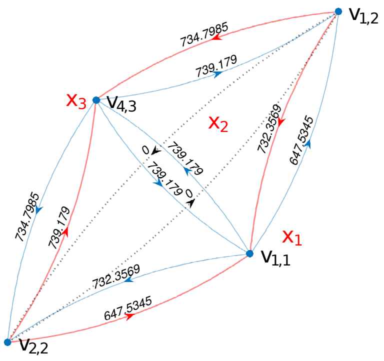
(b) Input graph



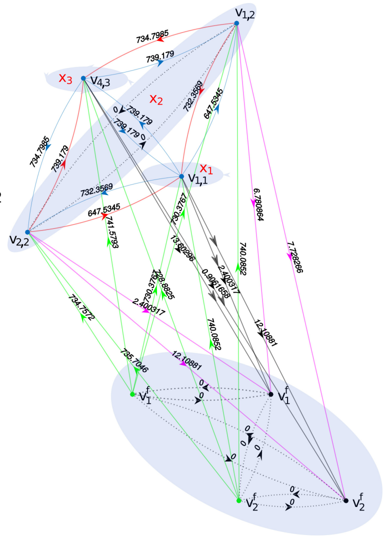
(c) Redundant nodes removed



(d) Duplicate nodes added to create non-intersecting clusters



(e) Noon-Bean transformation with penalty added and tail-shifted (red) edges.



(f) Robot depots added to the graph. Outgoing depot edges are shown in green, inter-cluster tail shifted edges are shown in red, incoming depot edges are shown in black solid, incoming tail-shifted depot edges are shown in magenta, intracluster zero cost edges are shown as dotted-black.

Figure 2.2: Transforming the multi-robot watchmen route problem into an asymmetric TSP instance.

6. This gives us a directed TSP input instance. This can be converted to an undirected TSP using transformation used by Karp [39].

The correctness of the algorithm follows from the correctness of the original Noon-Bean reduction [56]. The major change to the reduction is the addition of the complete graph between the depot vertices, $V_b$. The only incoming edges to the start depots, $v_i^d$, are from the finish depot vertices, $v_j^d$. Similarly, the only outgoing edges from the finish depot vertices are to the start depot vertices. Consequently, whenever the optimal TSP tour visits a finish depot vertex it must take a zero cost edge to a start depot vertex, from which it may either to a node in $V^t$ or to another finish depot vertex. Therefore, the TSP tour visits an alternating sequence of start and finish depot vertices with possibly non-zero viewpoints (*i.e.*, $V^t$ vertices) in between. We can, therefore, partition the TSP tour into $m$ subtours from start depot to finish depots. This gives us paths for the $m$ robots. One or more of these subtours may be empty, in which case the optimal solution uses fewer than $m$ robots. This can happen since the algorithm minimizes the sum of the path lengths and not the maximum path length of $m$ robots.

## 2.5 Evaluation

We implemented the algorithm described in the previous section. Our implementation is available online at `https://github.com/raaslab/watchman_route` and uses the Noon-Bean implementation from Mathew et al. [46] and VisiLibity library [57]. Figure 2.2 shows a 2D instance solved using this algorithm.

The penalties added to the edges can cause their cost to become large enough to run into numerical overflow issues. In our experiments, we encountered instances where the penalty resulted in the edge costs becoming large than what can be represented with the data structure used by Concorde. In such a case, we can use the reduction given by Lien and Ma [44] which does not require the addition of any penalty. However, their reduction triples the number of nodes in the TSP as compared to the Noon-Bean transformation. This results in larger instances and slower running times. Our single robot implementation based on the Lien-Ma transformation is also available online.

An instance with 15 targets and 30 candidate viewpoints for one robot took 41s secs to solve using Concorde, where as the same instance took 536s to solve directly using the Integer Linear Programming solver in MATLAB. We use an iterative implementation [36] to find the tour in MATLAB using the ILP function since specifying the full problem directly becomes too large to hold in memory.

Table 2.1 gives the times required to solve some representative problems using the Lien-Ma and Noon-Bean reductions with Concorde and MATLAB. As expected, the Noon-Bean reduction along with the Concorde solver is fastest among all options. The Lien-Ma reduction for an instance with 15 targets and 30 candidate viewpoints could not be solved in 12 hours using

| Reduction | Solver | Problem Size | Time (secs) |
|---|---|---|---|
| Lien-Ma | Concorde (DFS) | $\|V\| = 20, \|X\| = 10$ | 159.97 |
| Lien-Ma | Concorde (BFS) | $\|V\| = 20, \|X\| = 10$ | 225 |
| — | MATLAB ILP | $\|V\| = 20, \|X\| = 10$ | 40 |
| Noon-Bean | Concorde (BFS) | $\|V\| = 20, \|X\| = 10$ | 2.7 |

Table 2.1: Time required to solve a representative problem with various implementations.

Concorde. An instance with $|V| = 63$ and $|X| = 15$ could not be solved using MATLAB's ILP function in more than 16 hrs of computation. The same instance took 2411 secs with Concorde and Noon-Bean reduction. An instance with $|V| = 63$, $|X| = 15$, and three robots was solved in 1599 secs with Concorde and Noon-Bean.

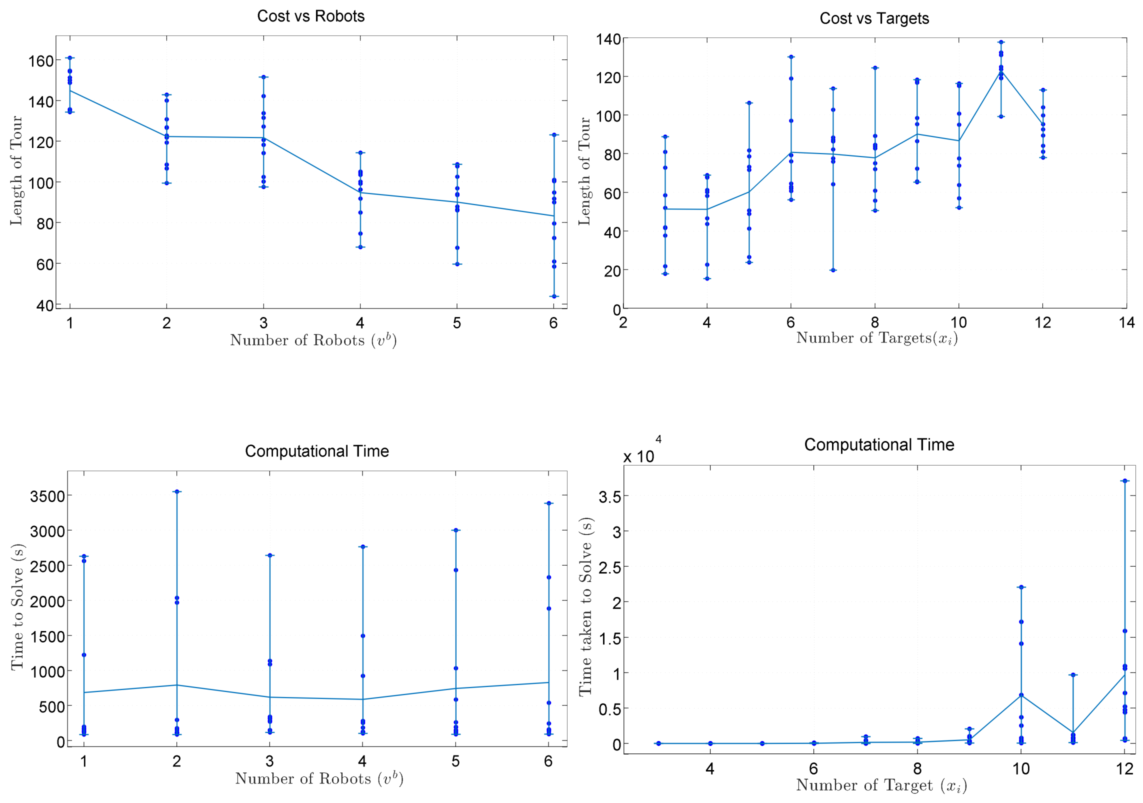


Figure 2.3: (a) Cost of the optimal solution with 10 targets as a function of the number of robots. (b) Cost of the optimal solution with 3 robots as a function of the number of targets. (c) & (d) Computational time for Figures (a) and (b), respectively. The dots show the costs/times of individual random trials and the curve shows the mean of all trials. The robots and target positions were randomly drawn in the environment shown in Figure 2.4.

| | Length of Tour Computed | Actual Distance Traveled |
|---|---|---|
| Robot 1 | 78.06 | 71.74 |
| Robot 2 | 50.04 | 43.63 |

Table 2.2: Comparison of the lengths of the tour computed and the actual distance traveled by the robots in the Gazebo simulation environment for Figure 2.4.

Figure 2.3(a) shows the effect of varying the number of robots on the optimal cost (sum of path lengths). 10 target locations are randomly generated for each trial in the environment shown in Figure 2.4. Figure 2.3(b) shows the effect of varying the number of targets. The number of robots was fixed to 3. Figures 2.3(c)–(d) show the computation time required for finding the solution using the Noon-Bean reduction with Concorde (BFS) solver, as a function of the number of robots and the targets.

The resulting algorithm was also tested in the Gazebo simulation environment (Figure 2.4) using two Pioneer 3DX robots fitted with a limited field-of-view angle camera. The robots emulate an omnidirectional camera by rotating in place whenever they reach a new vertex. Table 2.2 shows the comparison between the lengths of the tours on the input graph and the actual distance traveled by the robots in the Gazebo simulation environment. The actual distances are shorter since the robot is not restricted to move on the input graph in the polygonal environment.

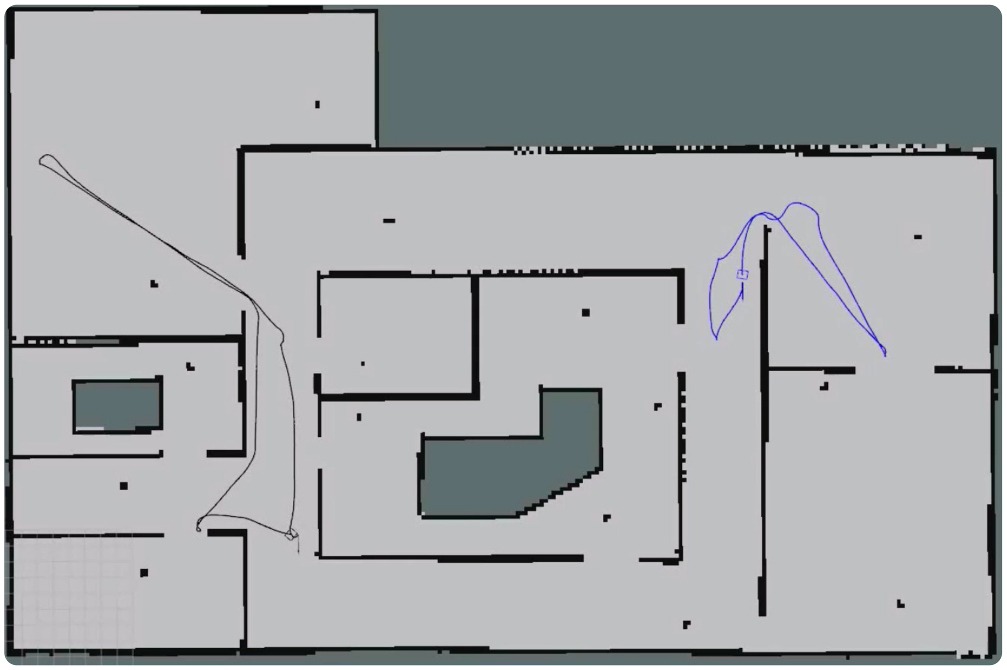


Figure 2.4: Two robot simulation in Gazebo. A video of the simulation is available online: `https://youtu.be/lvUyoFZBqxA`. The targets are marked as black dots.

# Chapter 3

# View Point Planning For Dynamic Scenes[1]

## 3.1 Introduction

Work in this chapter is motivated by applications where UAVs are used for dynamic environmental monitoring. Specifically, we are interested in tracking expanding plumes in aquatic environments using multiple robots. The hardware setup for this work is described in Chapter 4. We abstract the problem by defining points of interest as mobile targets we want to track with downward facing cameras on UAVs.

This work addresses the problem of assigning robots with limited Field-of-View (FoV) sensors to track multiple moving targets. We assume that each robot has a number of motion primitives to choose from. Motion primitive is the position a UAV can move to in one-time step. We can discretize the direction a UAV can move into required resolution. The assignment of targets to track is therefore coupled with the selection of motion primitives for each robot. We term this as the distributed Simultaneous Action and Target Assignment (SATA) problem.

A motion primitive is defined as a control input from the given state of a robot. We can generate a local trajectory by applying a sequence of motion primitives [30]. Although we use the term motion primitive throughout this thesis, we will interchangeable use it to refer to the local trajectories as well as the final state on these local trajectories if the usage is clear from the context. This naturally includes the collision avoidance as not to choose motion primitives that may encounter other robots or obstacles.

A motion primitive can track a target if at the end of the primitive the robot has the (predicted position of the) target in the FoV. Therefore, the set of targets tracked by different motion primitives for the robot may be different (Figure 3.1). Our goal is to assign motion primitives to the

[1]This work is part of the paper- Sung, Yoonchang and Budhiraja, Ashish Kumar and Williams, Ryan K and Tokekar, Pratap. Distributed Simultaneous Action and Target Assignment for Multi-Robot Multi-Target Tracking. arXiv preprint arXiv:1706.02245.

robots so as to track the most number of targets.

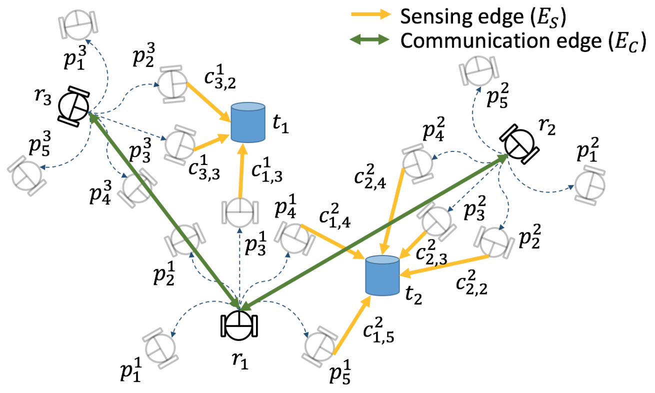


Figure 3.1: Description of multi-robot task allocation for multi-target tracking. In this illustration, the number of motion primitives for each robot is given by five.

This problem can be viewed as a set cover and its dual maximum cover [73] where every target is covered by a FoV of at least one motion primitive of a robot. It is equivalent to find a set cover of motion primitives of any robots with the minimum size. However, we have the additional constraint that only one motion primitive per robot can be chosen at each step. This implies that the relationship between a robot and the corresponding motion primitives turns out to be a packing problem [73] where only one motion primitive can be "packed" per robot. The combination of two aforementioned problems is called a Mixed Packing and Covering Problem (MPCP) [86].

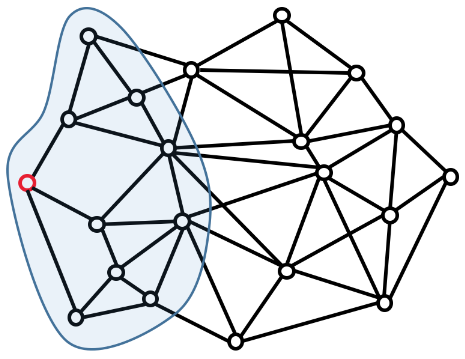


Figure 3.2: Communication graph of an anonymous network. The blue area indicates a radius-2 neighborhood of the red node. The red node is unaware of the entire network topology. A local algorithm that works for the red node only requires a local information of nodes in the blue area. The same local algorithm is applied to all nodes in the network to achieve a global goal of a given task.

## 3.2 Related Work

The problem of target tracking in dynamic scenes can be viewed as MPCP. There are many works in the literature that are studying the problem in this form. Young et. al. [86] provides a approximate solution to the problem. The time complexity of the algorithm is polylogarithmic in the input size. Jain et. al. [37] provide a faster parallel approximation algorithm for a special class of MPCP problems called semidefinite programs. The work done by Azar et. al. [7] uses existing framework of solving linear programs online and extends it to solve MPCP problems online.

There have been many studies on cooperative target tracking in both control and robotics communities. We highlight some of the most closely related works in this section. A more comprehensive survey of multi-robot multi-target tracking see [40].

Charrow et al. [16] proposed approximate representations of the belief to design a control policy for multiple robots to track one mobile target. The proposed scheme uses a centralized approach. Yu et al. [87] worked on an auction-based decentralized algorithm for cooperative path planning to track a moving target. Capitan et al. [12] proposed a decentralized cooperative multi-robot algorithm using auctioned partially observable Markov decision processes.

Morbidi et al. [52] studied a gradient-based control scheme for active multi-target tracking. They study both cooperative and non-cooperative aspects of the problem. Ahmad et al. [2] proposed a least squares minimization technique for cooperative multi-target tracking. However, they focus on localization, not on the multi-robot multi-target assignment.

Tokekar et. al. [77] prove that maximum numbers of target covered by selecting the robot trajectories greedily is 1/2 the optimal. Banfi et. al. [8] introduce new constraints in the optimization which take fairness of tracking into account. Zhang et. al. [90] divide the problem into a two level problem: at the top, robots use ranking and aggregation technique to allocate the target to each observer and at the lower level each observer acts independently to track a target. Recent work by Adamey et. al. [1] solves the problem by developing a data structure called region allocation tree and then using a recursive processing strategy.

The works in [41], [9] and [22] proposed algorithms to solve simultaneous task allocation and path planning, similar to SATA. They don't consider the `WinnerTakesAll` version of the problem.

## 3.3 Problem Descrition

Let $R$ be a set of robots and $T$ be a set of targets. Considering that $|R|$ robots are tracking $|T|$ targets, $R(k) = \{\mathbf{r}_1(k), ..., \mathbf{r}_i(k), ..., \mathbf{r}_{|R|}(k)\}$ denotes the state of robots at time $k$ and $T(k) = \{\mathbf{t}_1(k), ..., \mathbf{t}_j(k), ..., \mathbf{t}_{|T|}(k)\}$ denote the (estimated) state of targets at time $k$, where $\chi^R \subseteq \mathrm{Re}^n$ and $\chi^T \subseteq \mathrm{Re}^m$ are robot state and target state spaces, respectively. It is assumed that targets can be uniquely detected by a sensor attached to robots so that multiple robots know whether or not the same target is observed. This assumption excludes the necessity of data association over multiple robots, which is not the scope of this work.

As shown in Figure 3.1, each robot is able to compute feasible motion primitives of its own and detect multiple unique targets within the FoV. Then, the objective of the proposed problem is to choose one of motion primitives for each robot, yielding either the maximum number of targets being tracked by the robots or the best quality of tracking, depending on the application. One possible quality of tracking can be measured by the summation of all distances between selected primitives and the observed targets.

We first show how to formulate our problem as an Integer Program (IP). We define two unknown binary variables: $x_m^i$ and $y_i^j$. $x_m^i$ represents $i$-th robot selecting $m$-th motion primitive defined by $x_m^i = 1$ if $\mathbf{p}_m^i$ is selected by $\mathbf{r}_i$ and 0 otherwise. $y_i^j$ represents $i$-th robot assigned to track $j$-th target defined by $y_i^j = 1$ if $\mathbf{r}_i$ is assigned to $\mathbf{t}_j$. It follows:

$$\begin{aligned} \sum_{\mathbf{p}_m^i \in P^i} x_m^i \leq 1 \;\; \forall \mathbf{r}_i \in R, \\ \sum_{\mathbf{r}_i \in R} y_i^j \leq 1 \;\; \forall \mathbf{t}_j \in T. \end{aligned} \tag{3.1}$$

The objective is to maximize the number of targets that are tracked (alternatively, quality of tracking):

$$\underset{x_m^i, y_i^j}{\arg\max} \sum_{\mathbf{t}_j \in T} \left( \sum_{\mathbf{r}_i \in R} y_i^j \left( \sum_{\mathbf{p}_m^i \in P^i} c_{i,m}^j x_m^i \right) \right), \tag{3.2}$$

where $c_{i,m}^j$ denotes weights on sensing edges $E_S$ between $m$-th motion primitive of $i$-th robot and $j$-th target. $c_{i,m}^j$ can represent, for example, the distance between $t_j$ and $p_m^i$. Alternatively, $c_{i,m}^j \in \{0,1\}$ making the objective function equal to the number of targets tracked. Consequently, an optimal motion primitive $\mathbf{p}_m^{i*}$ for all robots can be selected based on $x_m^i$ and $y_i^j$. We term this as the WINNERTAKESALL version of SATA.

The SATA problem is NP-Hard [83]. The WINNERTAKESALL version can be optimally solved using a Quadratic Mixed Integer Linear Programming (QMILP) solver in the centralized setting.

## 3.4 Experiments

We performed a comparison study between QMILP and the local algorithm proposed by Sung et. al. [72]. QMILP solves for the optimal baseline objective that is WINNERTAKESALL and local algorithm solves for the BOTTLENECK objective. However, we compare the total number of targets covered by both approaches. We used TOMLAB [79] to solve the QMILP problem. The toolbox works with MATLAB and uses IBM's CPLEX optimizer in the background. On a laptop with processor configuration of Intel Core i7-5500U CPU @ 2.40GHz x 4 and memory 16 GB the maximum time to solve is around 4 seconds on a case with 200 targets and target average degree 2. Most of our cases were solved in less than 2 seconds.

We randomly generated graphs similar to Figure 3.3 with a given average degree for the comparison. We start with the upper half of the graph, connecting each robot to its two motion primitives. Then we iterate through each of motion primitive and randomly choose a target node to create an edge. Next, we iterate through target nodes and randomly choose a motion primitive to create an edge. We also add random edges to connect disconnected components (to keep the implementation simpler). We repeat this in order to get the required graph. If we need to increase the degree of target nodes in the graph, we create new edges to random primitives till we achieve the desired degree. We generated cases by varying the degree of targets, the number of targets, and the number of robots using the method described above.

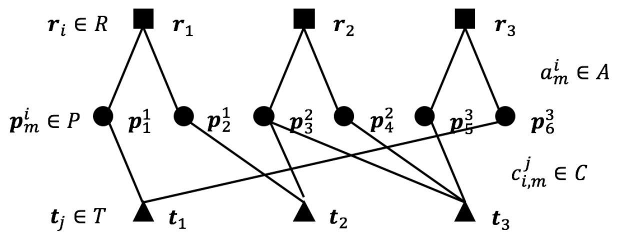


Figure 3.3: One instance of a graph for MPCP when there are three robot nodes, six motion primitive nodes and three target nodes.

The comparative simulation results are presented in Figure 3.4. The plots show minimum, maximum, and average of the targets covered by the local algorithm and QMILP running 10 random instances for every setting of the parameters. We also show the number of targets covered when choosing motion primitives randomly as a baseline. We observe that the local algorithm proposed by Sung et. al. [72] with $h = 2$ performs comparatively to the optimal algorithm and is always better than the baseline. In all the figures, $\Delta_R = 2$, making random a relatively powerful baseline.

In the case where the number of targets is 50 and 100 with degree 4 in Figure 3.4, the performance of the local algorithm does not improve as the number of robots deployed increases, which may seem counter-intuitive. We conjecture that the reason behind this is the locality of the proposed algorithm. Even though more robots are used to track the same number of targets, the average degree of the target remains the same. Consequently, the communication graph for the robots becomes sparser. Since $h$ is fixed for all cases, this implies that each robots layered graph reaches a smaller subset of the total graph, leading to even more sub-optimal performance. One avenue of future work is to analyze this in more depth.

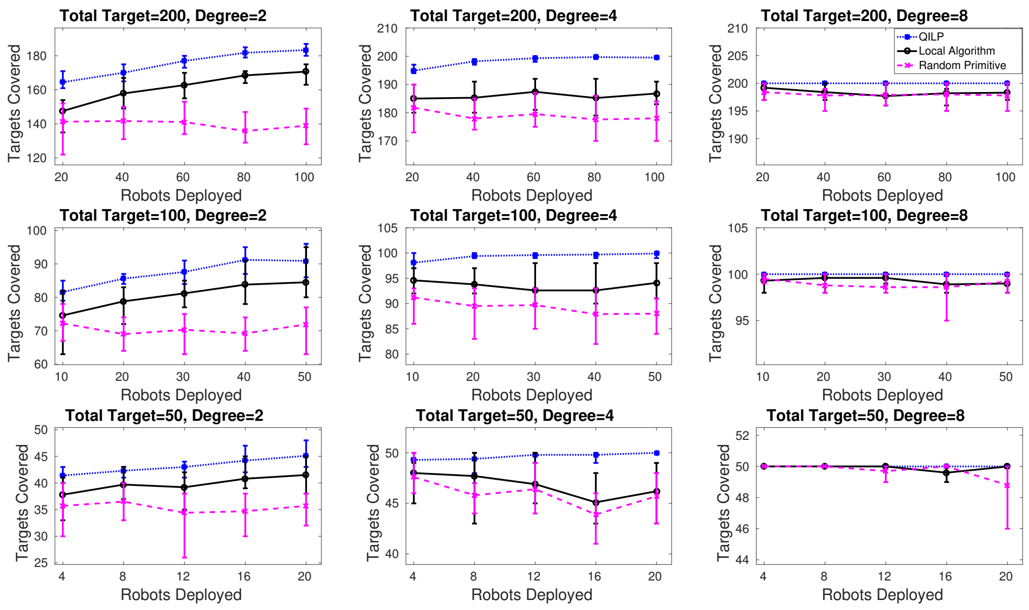


Figure 3.4: Showing the comparative results of QILP, proposed local algorithm by Sung et. al. [72], and randomly choosing a motion primitive. To generate graph we vary the number of robots, the total number of targets, and the average degree of a target.

## 3.5 Conclusion

This work gives an optimal baseline to solve the multi-robot multi-target assignment problem for any local algorithm. We want to address the scenarios where the robots would like to reduce their communication to solve the given assignment problem while at the same time maintaining some guarantees of tracking. Sung et. al. [72] worked on a powerful local communication framework employed by Floreen et al. [24] to leverage an algorithm that can trade-off the optimality with communication complexity. We empirically evaluated this algorithm and compared it with the baseline greedy strategies. Our immediate future work is to solve the WINNERTAKESALL version of SATA and extend the QMILP framework to work in a decentralized way. We can also add communication round between robots as another decision variable. We are also working on implementing the resulting algorithms on actual aerial robotic systems to carry out real-world experimentation. The preliminary field tests and hardware configuration are described in Chapter 4.

# Chapter 4

# System and Calibration

In this chapter, we present the hardware and software design of the UAV platform developed to support the two motivating applications. We also present results from preliminary field experiments.

## 4.1 Design Requirements

The main requirements for the UAV platform were flexibility to add custom payloads and upgrading the onboard sensors. The autopilot must be capable of operating in GPS-denied environments (in particular, for the bridge inspection application). Furthermore, the autopilot must be able to support programming with Robot Operating System [66]. The platform must have high endurance (>30 mins) because it can be time-consuming to retrieve and relaunch in the environments we want to deploy our UAVs. We wanted a platform that can handle in-flight communication between UAVs. The machine vision camera must give us the control over fixing focus, exposure, aperture and frame rate. Control over exposure is particularly important for the lighting conditions in the operating environment (for both applications) can change dramatically. The camera must also be a global shutter one, since rolling shutter on a UAV may lead to motion blurs. To enable onboard image processing, we need an onboard computer with computing capabilities at par with consumer-grade laptops. In Chapter 4.2 we talk about how our system can handle the above-mentioned requirements.

## 4.2 System Components

We choose the DJI-450 frame (Figure 4.1) as the base for our design. The frame has a diagonal wheelbase of 450mm. This frame supports 12-inch propellers and has enough space to mount an Intel NUC [75]. Components of the UAV are shown in Figure 4.2. This frame makes it easy to mount other peripherals such as the onboard camera and computer. We use 3D printed parts to attach these peripherals (see Figure 4.3).

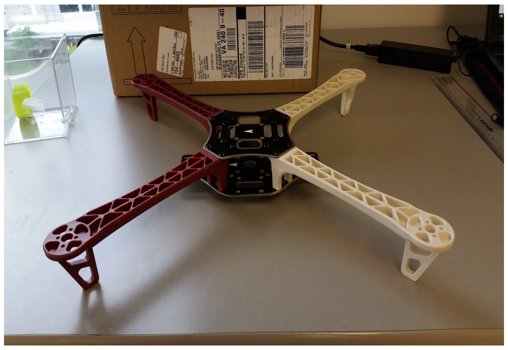


Figure 4.1: DJI-450 frame for our UAV.

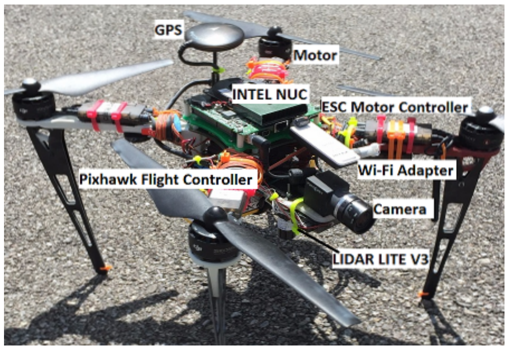


Figure 4.2: Labeled components of our UAV.

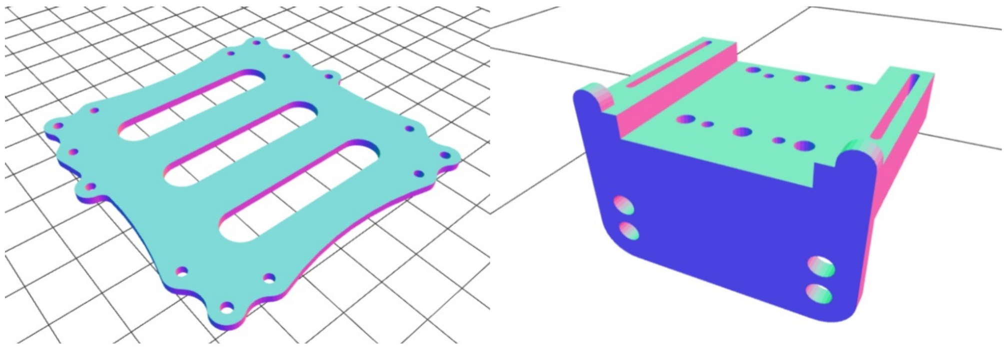


Figure 4.3: Shows in-house 3d printed components to mount camera and Intel NUC computer.

### 4.2.1 Autopilot and Propulsion

**PixHawk flight controller:** We choose PixHawk [48] autopilot for our platform. PixHawk is a high-performance autopilot-on-module suitable for fixed wing, multi rotors, and helicopters as well as many other robotic platforms. It is targeted towards high-end research, amateur, and industry needs. The Pixhawk platform is commonly used by many existing systems and is well-supported. PixHawk flight controller is shown in Figure 4.4.

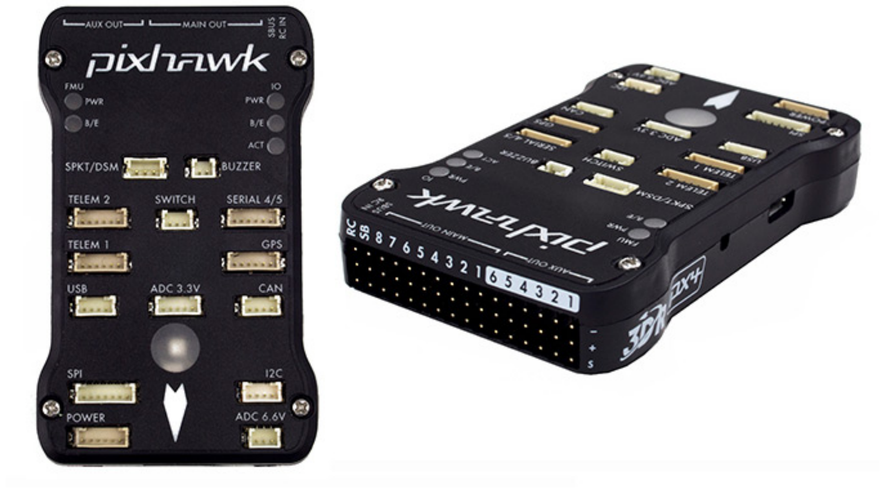


Figure 4.4: PixHawk Flight Controller. Image credits: [33].

**GPS:** We use the 3DR GPS module that has a u-blox NEO-7 module inside. The GPS provides data at 5Hz and weighs 16.8 g with the casing. Though PixHawk autopilot has its internal compass the one inside this module is more reliable.

**Motor and motor controllers:** We use the E800 Tuned Propulsion System [21] manufactured by DJI in our system, as shown in Figure 4.5. This is the component of our system that provides it the endurance we need. Each motor supports a 12-inch propeller and can provide a maximum thrust of 1400 g. At hover, each motor provides around 600 g thrust. The overall weight of our system is 2.4 kg. The current needed by the whole system at this thrust is approximately 12 A. This efficiency in the system is mainly due to low Kv rating (350 Kv), single strand coils and most importantly switching using sinusoidal back EMF.

Figure 4.5: E800 Tuned Propulsion System by DJI. Image credits: [21].

Figure 4.6: Tattu 8000mAh 22.2V 25C 6S1P lipo Battery pack used for UAVs. Image credits: [35].

**Battery:** Another important factor in increasing the endurance is the battery. We used Tattu 8000mAh 22.2V 25C 6S1P Lipo Battery Pack as shown in Figure 4.6. The weight of this battery is 1160 gms. The final endurance is approximately 40 mins.

### 4.2.2 Camera

We use the PointGrey Flea3 [23] FL3-U3-20E4C-C camera for our system. The camera (see Figure 4.7) is mounted side-looking for bridge inspection and downwards-facing for plume tracking. The resolution of the images is $1600 \times 1200$. The camera supports frame rates up to 59 frames per second. The camera uses sensor e2v EV76C5706F CMOS with a global shutter. The lens used is shown in Figure 4.8. The lens provides wide viewing angle of 50.8 x 38.6° for 1/1.8" megapixel cameras. It has an adjustable aperture range $f/1.4-16$ and fixed focal length $8mm$.

Figure 4.7: Flea3 2.0 MP Color USB3 Vision - FL3-U3-20E4C-C. Image credits: [31].

Figure 4.8: Tamron M118FM08 Mega-Pixel Fixed-Focal Industrial Lens. Image credits: [34].

### 4.2.3 Onboard Computing

We use the Intel NUC7i7BNH Mini PC NUC Kit as shown in Figure 4.9, as our onboard computer. This mini computer has 3.5 GHz Intel Core i7-7567U Dual-Core processor. We can run high-level image processing algorithms required for our application on this processor. This supports RAM up to 32GB and without the metal casing (see Figure 4.10) weighs around 100 gm. This weight to processing power ratio is appropriate for our application.

Figure 4.9: Intel NUC NUC7i7BNH. Image Credits: [32].

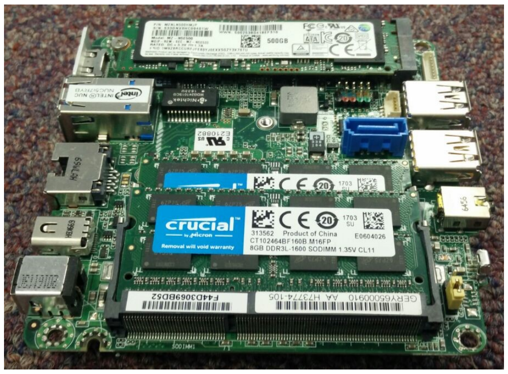


Figure 4.10: Intel NUC with metal casing removed

## 4.3 IMU-Camera Calibration

Our payload is a machine vision camera. Our eventual goal for the problem discussed in Chapter 3 is to estimate the position of targets using noisy measurements from multiple UAVs. To share the measurements amongst UAVs, we need to have a centrally agreed frame of reference. The camera gives measurements relative to camera frame. We need to transform these measurements to a global frame. This requires two calibration matrices. First, intrinsic camera matrix that yields focal length, optical center, skew, and distortion. These parameters are needed to convert an image point to camera frame [47]. Second, we need a static transformation matrix between the camera frame and the autopilot, i.e., the IMU frame. The transformation is static since the IMU and camera are rigidly mounted on the UAVs. See Figure 4.11 for a visualization of the coordinate frames involved. If we know both matrices then we can express the measurement in a global frame since we know position and orientation of IMU in a global frame from a sensor fusion algorithm that runs inside PixHawk autopilot [48] autopilot on our UAV. The fusion algorithm estimates the attitude of the UAV using accelerometer, gyroscope, and magnetometer. It also estimates the position of the UAV using GPS and Barometer.

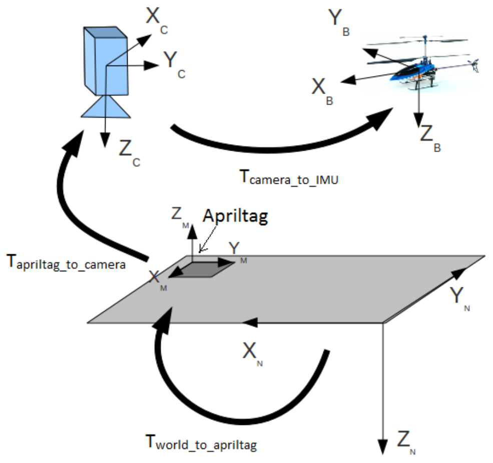


Figure 4.11: We determine $T_{camera\_to\_IMU}$ through Kalibr toolbox. This is a transform that contains translation and rotation of IMU frame w.r.t camera frame. Image taken from PixHawk project [63]

Determining the accuracy of calibration when integrated with the whole system was not possible especially with GPS in the loop. This can be attributed to the accuracy provided by GPS. The part we want to verify is the IMU to camera transform on an actual test bed. To test this we designed our own experiment where we used fake_gps [81] under motion capture to remove the inaccuracies induced from GPS and know how our calibration is behaving as a standalone. We talk more about this in Chapter 4.3.1.

### 4.3.1 Measurement

As part of the preliminary testing for tracking over aquatic bodies project in Chapter 3, we use Apriltag markers attached to foam buoys as a proxy for the expanding plumes (see Figure 4.12). Given the size of Apriltag marker, and the camera intrinsic parameters, an Apriltag detection [58] gives us the position and orientation of the tag with respect to the camera optical center. As discussed earlier we need a transformation from IMU frame to camera frame to eventually give a global measurement of an Apriltag see Figure 4.11. Below we explain the toolbox that we use, the data collection method and our test bed to verify calibration.

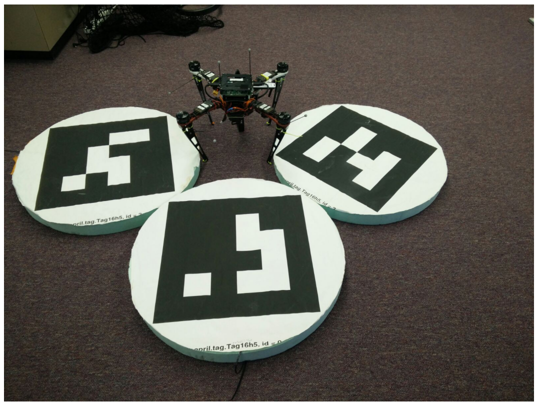


Figure 4.12: Showing Apriltag on foam buoys.

### 4.3.2 Toolbox

Kalibr [38] is a toolbox that solves for intrinsic and extrinsic calibration of multiple cameras and spatial and temporal calibration of an IMU w.r.t. a camera system. In an extended version, spatial and temporal calibration of a camera system w.r.t. multiple IMUs, as well as IMU intrinsic calibration, is supported. To make the calibration task more convenient and reproducible, camera focus and calibration validator are also part of the toolbox. The calibration approaches used in Kalibr are based on papers by Furgale et. al. [26] [27].

### 4.3.3 Data Collection

Input to the toolbox are three files:

1. **IMU.yaml**: contains sampling rate and constants for IMU noise model given in Eq. 4.1.

$$\tilde{\omega}(t) = \omega(t) + b(t) + \eta(t). \tag{4.1}$$

The constants are gyroscope noise density, accelerometer noise density, gyroscope random walk and accelerometer random walk. We can find the noise density parameter usually from datasheet and the random walk parameters can be determined directly from Allan standard deviation plot.

2. **Camera.yaml**: contains intrinsic parameters of the camera, we use Matlab camera calibration toolbox to find these intrinsic parameters. The toolbox is based on Zhang et. al. [91]. The pinhole camera model as shown in the Figure 4.13 used by the toolbox. The corresponding equations are given in Eq.4.2

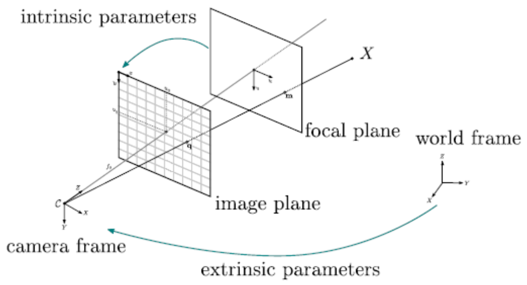


Figure 4.13: The pinhole camera model. An oriented central projective camera. Image Credits: [59]

$$P_{3\times4} = K[R|t] = \begin{bmatrix} f*k_u & & c_u \\ & f*k_v & c_v \\ & & 1 \end{bmatrix} \begin{bmatrix} & & & t_x \\ & R_{3\times3} & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

The description is taken from openmvg docs [59]. Eq. 4.2 the parameters are given as:

- Intrinsic parameters $[f; cu; cv]$:
  - $ku, kv$ : scale factor relating pixels to distance (often equal to 1),
  - $f$ : the focal distance (distance between focal and image plane),
  - $cu, cv$ : the principal point, which would be ideally in the center of the image.
- $R$ : the rotation of the camera to the world frame,
- $t$ : the translation of the camera. t is not the position of the camera. It is the position of the origin of the world coordinate system expressed in coordinates of the camera-centered coordinate system.

A 3D point is projected in an image with the following formula (homogeneous coordinates) in Eq. 4.3:

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \begin{bmatrix} f * k_u & & c_u \\ & f * k_v & c_v \\ & & 1 \end{bmatrix} \begin{bmatrix} & & & t_x \\ & R_{3\times3} & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ W_i \end{bmatrix} \tag{4.3}$$

3. **Rosbag**: recording containing IMU data and camera images. Rosbag is a set of tools for recording from and playing back to Robot Operating System (ROS) topics. It is intended to be high performance and avoids deserialization and reserialization of the messages.

### 4.3.4 Kalibr Algorithm and Output

The toolbox uses basis function approach for batch continuous-time state estimation as presented by Furgale et. al. [26]. Time-varying states are represented as the weighted sum of a finite number of known analytical basis functions. Using the estimated states they write the measurement model equations for the sensors involved namely camera, accelerometer and gyroscope. Also, they model IMU biases by zero-mean white Gaussian processes. There are five quantities that need to be estimated namely, gravity direction, the transformation from camera to IMU, offset between camera time and IMU time, pose of IMU w.r.t. world frame, accelerometer and gyroscope biases. They use Levenberg-Marquardt (LM) [45] [43] algorithm, this is a standard bundle adjustment algorithm. The objective function in turn minimizes the errors for all the measurement and process models involved. The toolbox outputs a transformation from IMU to camera. And also provides time offset estimate, camera reprojection errors, gyroscope and accelerometer RMS error.

## 4.4 Calibration Verification

As discussed before we need a method to verify the quality of the calibration in the whole system. We use motion capture (MOCAP) (see Figure 4.14) system to provide a fake GPS [81] location to our PixHawk autopilot. We placed a grid of Apriltags in the MOCAP area on the floor (see Figure 4.15). We recorded a rosbag containing transform of these Apriltags and UAV using ROS package called tf [74]. Figure 4.16 shows a visualisation of the ROS computation graph. We provide the code for our implementation [25]. In order to analyze the data from the Rosbag we move it to Matlab Robotics Toolbox [67] where we plot the positions of the tags (Figure 4.17) in the MOCAP coordinate frame. We compare the ground truth of Apriltags distances with mean distances as shown in Figure 4.17 and in Table 4.1. To plot positions of the tag in MOCAP coordinate frame we use the IMU to camera transformation we found above. Since we know the true distances between the tags we are able to analyze the mean and spread of the measurement. While the error in the representative indoor trials is in the order of centimeters, a larger scale evaluation must be performed outdoors by increasing the altitude of the aerial vehicle.

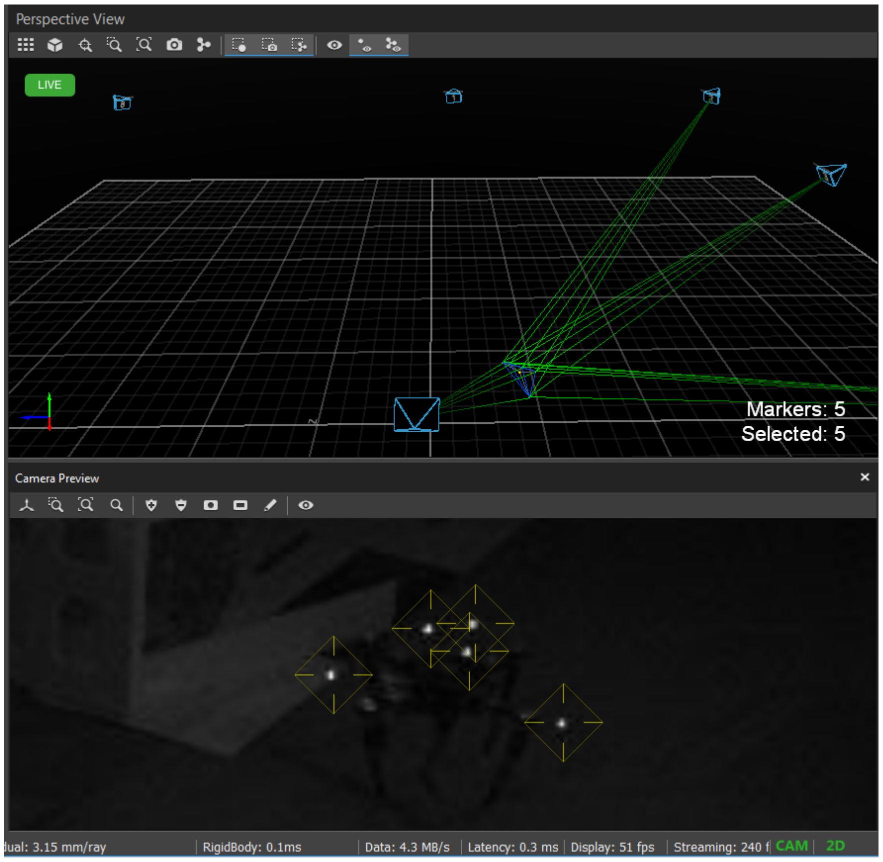


Figure 4.14: Screenshot showing a UAV with markers under MOCAP.

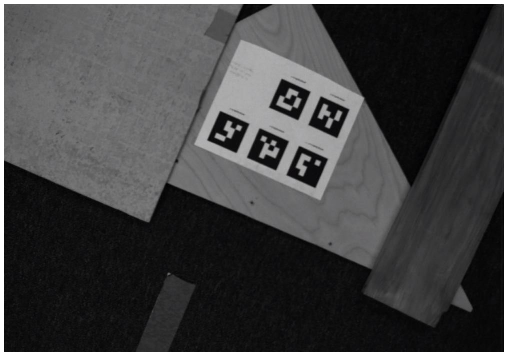


Figure 4.15: Image from UAV camera flying in MOCAP arena

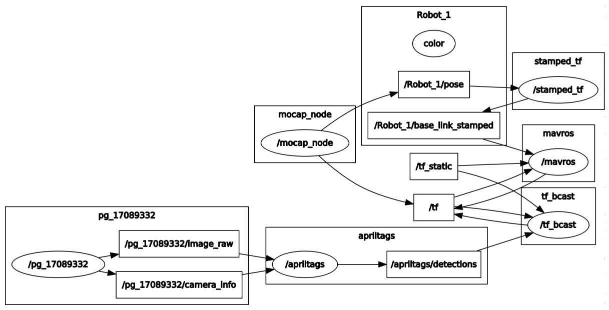


Figure 4.16: Rosgraph showing the active nodes. pg_17089332 is running the camera driver, mocap_node is publishing UAV's position in MOCAP frame, apritags node is handling the image processing task to read the tags, mavros node takes MOCAP position input and fakes it as GPS for PixHawk autipilot. Other nodes perform supporting roles.

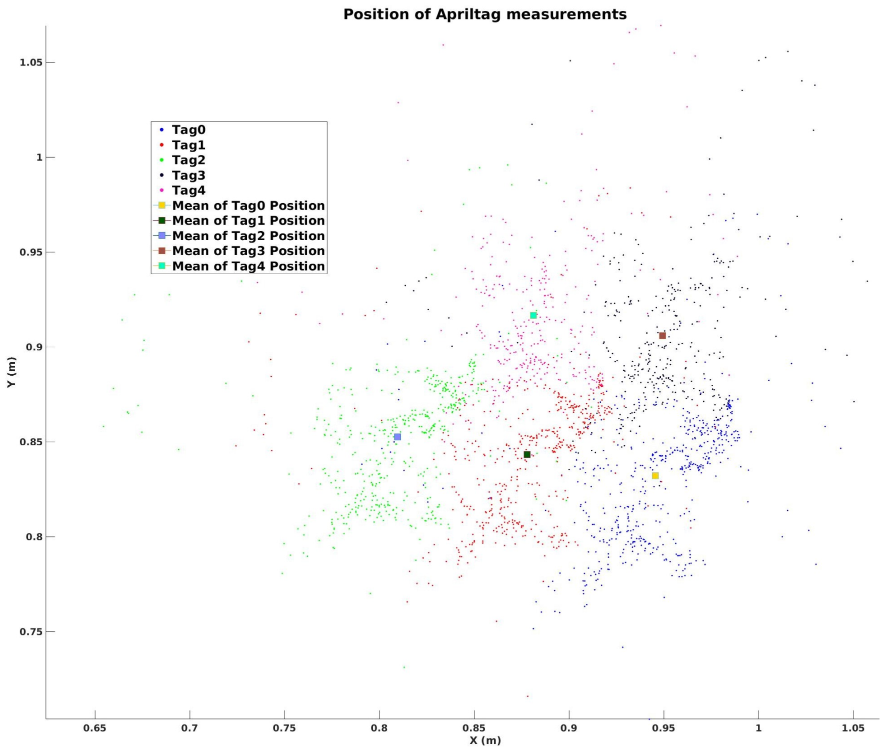


Figure 4.17: Shows position of Apriltag in MOCAP frame as seen by flying UAV.

| | Ground Truth Distance (m) | Mean Distance (m) |
|---|---|---|
| tag0 to tag1 | 0.0678 | 0.0684 |
| tag1 to tag2 | 0.0678 | 0.0690 |
| tag3 to tag4 | 0.0678 | 0.0690 |
| tag0 to tag3 | 0.0864 | 0.0734 |
| tag1 to tag4 | 0.0864 | 0.0737 |

Table 4.1: Comparing ground truth with distance between mean positions. MOCAP provides minimum resolution of 8 mm.

## 4.5 Preliminary Experiments

To analyze the challenges and understand the limitations of our UAVs for bridge inspection project we performed experiments at Virginia Tech Transport Institute's (VTTI) smart bridge [69] (see Figure 4.18) and George Coleman bridge [28] (see Figure 4.19). Through our experiments, we realized that the GPS based position hold is unreliable flying inside or close to the bridge. Figure 4.20 shows the UAV flying inside the smart bridge and George Coleman bridge. Heading correction based on compass was more unreliable inside the bridge compared to outside. Furthermore, owing to the more exposed metal in George Coleman bridge compared to VTTI's smart bridge, the compass is even more unreliable. UAVs experienced strong wind gusts flying inside or near the George Coleman bridge. This is because the openness of the structure causes wind channels between the trusses and being over a river causes excess winds. We flew the UAVs in manual mode or in altitude hold mode [65]. In manual mode, the UAV holds the attitude and we have to control the drift and thrust. In altitude hold mode we only have to control the drift and UAV holds the height using the barometer. We used the images captured from the outside of the bridge and used Pix4D [64] software to stitch them together to give us the reconstruction shown in Figure 4.21. Figure 4.22 shows one of an image taken by our UAV while inspecting the George Coleman bridge.

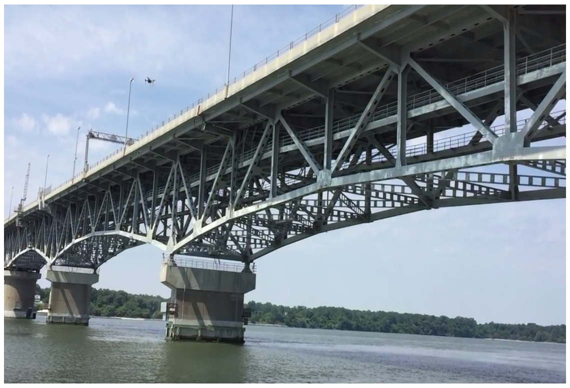


Figure 4.18: A shot capturing our UAV recording images from the side of the George Coleman bridge.

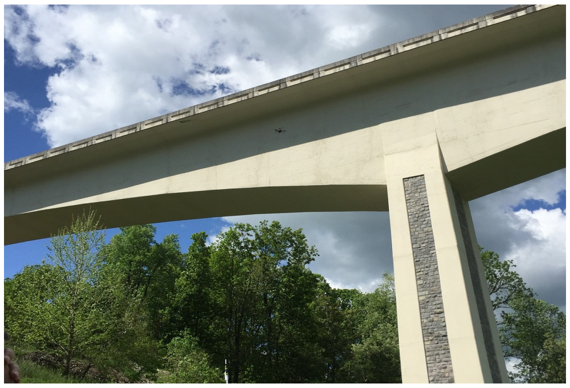
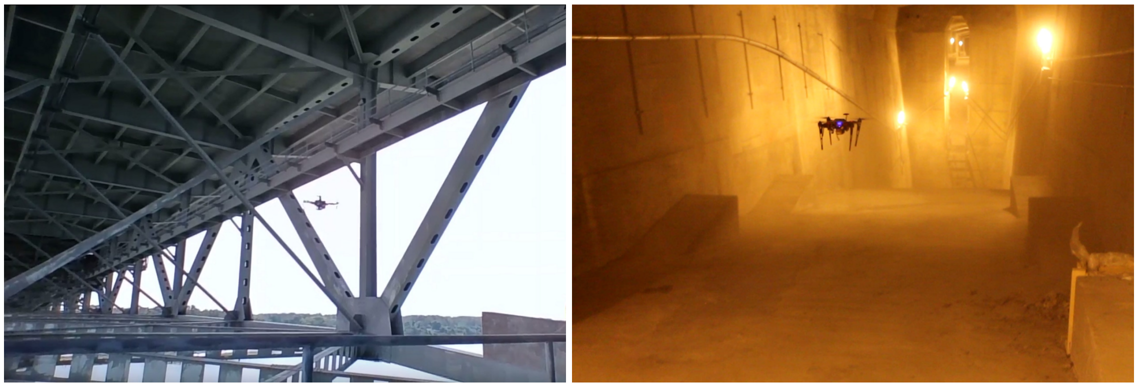


Figure 4.19: A view of VTTI's smart bridge with our UAV in the scene recording the images.



(a) Flight inside the George Coleman bridge (b) Flight inside VTTI's smart bridge

Figure 4.20: Manual flights inside the bridge to assess the bridge health from inside.

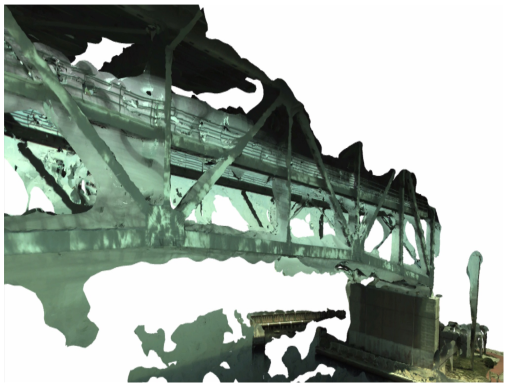


Figure 4.21: Shows reconstruction of the George Coleman bridge.

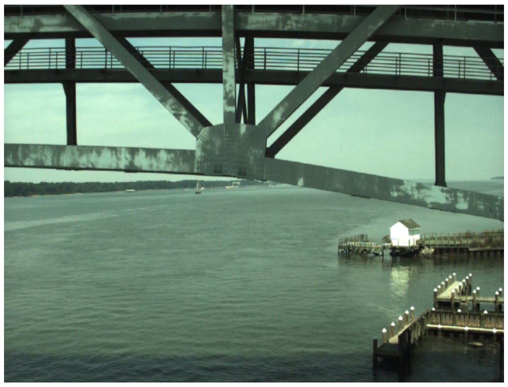


Figure 4.22: Image taken by the UAV while flying.

We also performed several experiments to test our setup and understand the problems of flying UAVs over aquatic environments. In the initial phases of the experiments, we tested the UAV's image capturing and image processing capabilities. In Figure 4.23 we can see an Apriltag being detected. We are able to record Apriltags measurements at 35 fps using a 59 fps camera. We used personalrobotics/apriltags [68] repository for the Apriltag's image processing.

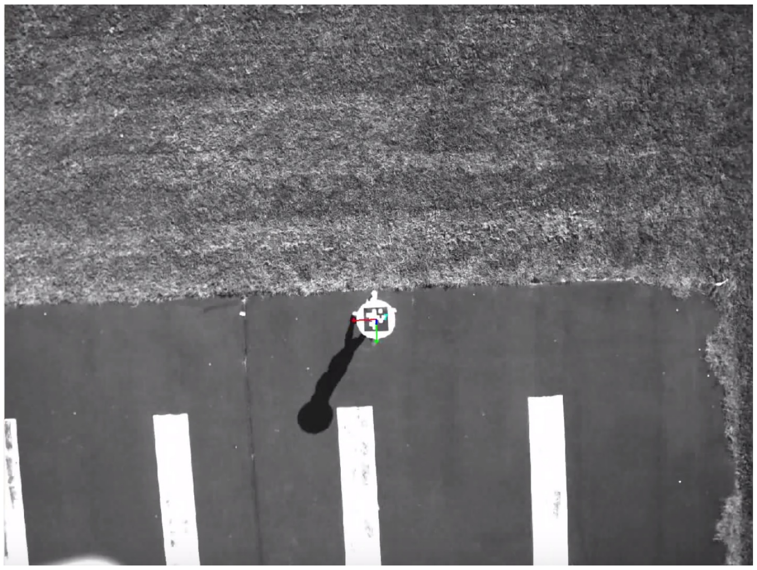


Figure 4.23: Apriltag detected using onboard image processing algorithms.

Next round of experiments was performed at Claytor lake in Virginia. The most challenging part of flying over the lake was not having water landing capabilities. We are currently working towards enabling our UAVs for a water landing. Another important requirement to capture good videos is the ability to control exposure. Water reflects a large amount of light on a sunny day, it is impossible to read tags with auto exposure enabled. We also recorded the data with Apriltag on a mannequin and an unmanned surface vehicle (USV) (See Figure 4.24). Figure 4.25 shows the UAV flying over the mannequin and the USV.

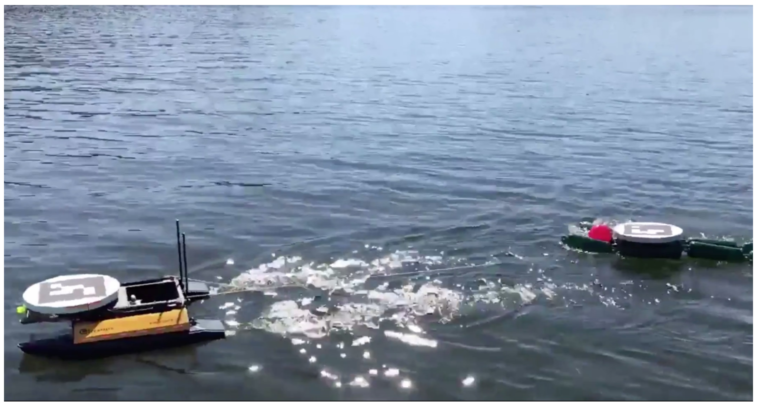


Figure 4.24: Apriltags mounted on a mannequin and a USV.

Figure 4.25: UAV recording the videos of the tags at Claytor lake.

# Chapter 5

# Conclusion

In this thesis, we study the problem of viewpoint planning in static and dynamic scenes using multi-robot teams. For static scenes with polygonal environments, the goal is to compute tours for multiple robots such that there is at least one tour that can see all the required targets in the environment. We presented a practical solution that finds an optimal solution to the problem in possibly exponential time. In the case of dynamic environments, we studied the multi-robot assignment problem for multi-target tracking. We solved the problem optimally using Mixed Quadratic Integer Linear Program to maximize the total number of targets covered. We also discussed our system design, experiments for bridge inspection and tracking plumes in aquatic environments.

The algorithm described in Chapter 2 finds the optimal solution by reducing it to GTSP. However, this is valid when the cost function is the sum of the travel times for all the robots. For this algorithm, we discretize the environment to get the candidate viewpoint nodes. Instead of naively discretizing, we can discretize only along the edges of the visibility polygon and add these discrete points to the graph. We would also need to add vertices of the polygon/holes to the graph. It can be proven that optimal tour that sees all the targets in the environment would pass through the vertices of the polygon/holes and/or through the edges of the visibility polygon. We leave this proof for future work. We also would like to extend this approach to account for measurement time. Minimizing the makespan remains an open problem. Adding measurement time can be done quite easily, we would need to add a measurement cost to each edge in $G^{cr}$ in Chapter 2.4 Step 2. The experimental evaluation also remains a part of our future work.

For dynamic viewpoint planning discussed in Chapter 3, we want to focus on scenarios where the number of robots is large and consequently solving the problem locally rather than centrally is desirable. The robots may have a limited communication range and bandwidth. As such, we seek assignment algorithms that rely on local information and limited, local communication with the neighboring robots. The eventual aim of this line of work is to maximize the total number of targets while decreasing the communication rounds between the robots.

In the decentralized problem framework, Floréen et al. [24] proposed a local algorithm to solve MPCP using max-min/min-max LPs in a distributed manner. A local algorithm is a constant-time distributed algorithm that is independent of the size of a network [73]. This enables a robot only to depend on local inputs in the constant-radius neighborhood of robots. Figure 3.2 illustrates the gist of local algorithm. The scalability can be achieved by employing a local algorithm, as no global information is required. In other words, each robot does not need to know the total number of robots collaborating with. The work to analyze dynamic scenes in this thesis acts as a baseline to such algorithms. Our preliminary results on adapting an existing local algorithm for tracking, performed along with Sung et al. [72], are promising.

Our immediate future work is an experimental validation of algorithms in Chapter 2 and 3 with the hardware system described in 4.

# Bibliography

[1] Emrah Adamey, Abdullah Ersan Oğuz, and Ümit Özgüner. Collaborative multi-msa multi-target tracking and surveillance: a divide & conquer method using region allocation trees. *Journal of Intelligent & Robotic Systems*, pages 1–15, 2017.

[2] Aamir Ahmad, Gian Diego Tipaldi, Pedro Lima, and Wolfram Burgard. Cooperative robot localization and target tracking based on least squares minimization. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5696–5701. IEEE, 2013.

[3] Soroush Alamdari, Elaheh Fata, and Stephen L Smith. Persistent monitoring in discrete environments: Minimizing the maximum weighted latency between observations. *The International Journal of Robotics Research*, 33(1):138–154, 2014.

[4] David Applegate, ROBERT Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver. *URL http://www. tsp. gatech. edu/concorde*, 2006.

[5] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study.* Princeton university press, 2011.

[6] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 2–11. IEEE, 1996.

[7] Yossi Azar, Umang Bhaskar, Lisa Fleischer, and Debmalya Panigrahi. Online mixed packing and covering. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 85–100. Society for Industrial and Applied Mathematics, 2013.

[8] Jacopo Banfi, Jérôme Guzzi, Alessandro Giusti, Luca Gambardella, and Gianni A Di Caro. Fair multi-target tracking in cooperative multi-robot systems. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5411–5418. IEEE, 2015.

[9] John Bellingham, Michael Tillerson, Arthur Richards, and Jonathan P How. Multi-task allocation and path planning for cooperating uavs. In *Cooperative control: models, applications and algorithms*, pages 23–41. Springer, 2003.

[10] Bridge inspection and assessment by stantec. http://www.stantec.com/content/dam/stantec/images/projects/0013/george-washington-bridge-inspection.jpg, 2017. [Online; accessed 18-July-2017].

[11] Bridge inspection using cranes. https://uavamerica.com/bridge-inspection/, 2017. [Online; accessed 18-July-2017].

[12] Jesus Capitan, Matthijs TJ Spaan, Luis Merino, and Anibal Ollero. Decentralized multi-robot cooperation with auctioned pomdps. *The International Journal of Robotics Research*, 32(6):650–671, 2013.

[13] Svante Carlsson, Håkan Jonsson, and Bengt J Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete & Computational Geometry*, 22(3):377–402, 1999.

[14] Christos G Cassandras, Xuchao Lin, and Xuchu Ding. An optimal control approach to the multi-agent persistent monitoring problem. *Automatic Control, IEEE Transactions on*, 58(4):947–961, 2013.

[15] Brodie Chan, Hong Guan, Jun Jo, and Michael Blumenstein. Towards uav-based bridge inspection systems: A review and an application perspective. *Structural Monitoring and Maintenance*, 2(3):283–300, 2015.

[16] Benjamin Charrow, Vijay Kumar, and Nathan Michael. Approximate representations for multi-robot control policies that maximize mutual information. *Autonomous Robots*, 37(4):383–400, 2014.

[17] Jie Chen, Junjie Wu, Gang Chen, Wei Dong, and Xinjun Sheng. Design and development of a multi-rotor unmanned aerial vehicle system for bridge inspection. In *International Conference on Intelligent Robotics and Applications*, pages 498–510. Springer, 2016.

[18] Wei-pang Chin and Simeon Ntafos. Optimum watchman routes. *Information Processing Letters*, 28(1):39–44, 1988.

[19] Companies that make drones and drone accessories. https://uavcoach.com/drone-companies/#Hardware/, 2017. [Online; accessed 13-July-2017].

[20] Ajay Deshpande, Taejung Kim, Erik Demaine, and Sanjay Sarma. A pseudopolynomial time o (log n)-approximation algorithm for art gallery problems. *Algorithms and Data Structures*, pages 163–174, 2007.

[21] Dji's tuned propusion system e800. http://www.dji.com/e800. Accessed: July'19 2017.

[22] Yeonju Eun and Hyochoong Bang. Cooperative task assignment/path planning of multiple unmanned aerial vehicles using genetic algorithm. *Journal of aircraft*, 46(1):338–343, 2009.

[23] Flea3 2.0 mp color firewire 1394b (sony icx274). https://www.ptgrey.com/flea3-2-mp-color-firewire-1394b-sony-icx274-camera. Accessed: 2017-01-03.

[24] Patrik Floréen, Marja Hassinen, Joel Kaasinen, Petteri Kaski, Topi Musto, and Jukka Suomela. Local approximability of max-min and min-max linear programs. *Theory of Computing Systems*, 49(4):672–697, 2011.

[25] Code for reading the apriltags in global frame with a camera mounted on UAV. The UAV gets GPS using it's position under MOCAP system. `https://github.com/raaslab/Kalibr_Mocap`, 2017. [Online; accessed 3-Aug-2017].

[26] Paul Furgale, Timothy D Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2088–2095. IEEE, 2012.

[27] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1280–1286. IEEE, 2013.

[28] George p. coleman memorial bridge is a double swing bridge that spans the york river. `https://en.wikipedia.org/wiki/George_P._Coleman_Memorial_Bridge`. Accessed: July'19 2017.

[29] How do you clean up an oil spill? `http://science.howstuffworks.com/environmental/green-science/cleaning-oil-spill.htm`, 2017. [Online; accessed 13-July-2017].

[30] Thomas Howard, Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Model-predictive motion planning: several key developments for autonomous mobile robots. *IEEE Robotics & Automation Magazine*, 21(1):64–73, 2014.

[31] Image credits for flea3 camera image. `http://www.trossenrobotics.com/flea3`. Accessed: July'19 2017.

[32] Image credits for intel nuc image. `http://nucblog.net/2017/04/kaby-lake-i7-nuc-review/`. Accessed: July'19 2017.

[33] Image credits for pixhawk image. `http://www.fpvpro.com/dragonlink/support/v3-advanced/pixhawktelemetrysetup/`. Accessed: July'19 2017.

[34] Image credits for tamron lens image. `https://www.bhphotovideo.com/c/product/725574-REG/Tamron_M118FM08_M118FM08_Mega_Pixel_Fixed_Focal_Industrial.html`. Accessed: July'19 2017.

[35] Image credits for tattu battery image. `http://www.genstattu.com/tattu-8000mah-22-2v-25c-6s1p-lipo-battery-pack-with-xt60-plug.html`. Accessed: July'19 2017.

[36] Iterative integer linear programming method for travelling salesman problem. `https://www.mathworks.com/help/optim/ug/travelling-salesman-problem.html`. Accessed: 2016-11-04.

[37] Rahul Jain and Penghui Yao. A parallel approximation algorithm for mixed packing and covering semidefinite programs. *arXiv preprint arXiv:1201.6090*, 2012.

[38] Kalibr toolbox. `https://github.com/ethz-asl/kalibr/wiki`, 2017. [Online; accessed 14-July-2017].

[39] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[40] Asif Khan, Bernhard Rinner, and Andrea Cavallaro. Cooperative robots to observe moving targets: Review. *IEEE Transactions on Cybernetics*, 2016.

[41] AK Kulatunga, DK Liu, G Dissanayake, and SB Siyambalapitiya. Ant colony optimization based simultaneous task allocation and path planning of autonomous vehicles. In *Cybernetics and Intelligent Systems, 2006 IEEE Conference on*, pages 1–6. IEEE, 2006.

[42] Xiaodong Lan and Mac Schwager. Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2415–2420. IEEE, 2013.

[43] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[44] Yao Nan Lien, Eva Ma, and Benjamin W S Wah. Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem. *Information Sciences*, 74(1-2):177–189, 1993.

[45] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[46] Neil Mathew, Stephen L Smith, and Steven L Waslander. Multirobot rendezvous planning for recharging in persistent tasks. *IEEE Transactions on Robotics*, 31(1):128–142, 2015.

[47] Mathwork article on - what is camera calibration? `https://www.mathworks.com/help/vision/ug/camera-calibration.html`. Accessed: 2017-18-07.

[48] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, pages 1–19. 10.1007/s10514-012-9281-4.

[49] Najib Metni and Tarek Hamel. A uav for bridge inspection: Visual servoing control law with orientation limits. *Automation in construction*, 17(1):3–10, 2007.

[50] Nathan Michael, Ethan Stump, and Kartik Mohta. Persistent surveillance with a team of mavs. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2708–2714. IEEE, 2011.

[51] Joseph SB Mitchell. Approximating watchman routes. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 844–855. SIAM, 2013.

[52] Fabio Morbidi and Gian Luca Mariottini. Active target tracking and cooperative localization for teams of aerial vehicles. *IEEE Transactions on Control Systems Technology*, 21(5):1694–1707, 2013.

[53] Robin R Murphy. *Disaster robotics*. MIT press, 2014.

[54] National oceanic and atmospheric administration's (noaa) article on cleaning the oil spills. `http://response.restoration.noaa.gov/oil-and-chemical-spills/oil-spills/spill-containment-methods.html`, 2017. [Online; accessed 19-July-2017].

[55] News coverage on bridge inspection gone wrong by cbc news. `http://www.cbc.ca/news/routine-bridge-inspection-goes-terribly-wrong-1.3205166`, 2017. [Online; accessed 18-July-2017].

[56] Charles E Noon and James C Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, 31(1):39, 1993.

[57] K. J. Obermeyer and Contributors. The VisiLibity library. `http://www.VisiLibity.org`, 2008. R-1.

[58] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.

[59] Openmvg (multiple view geometry) is a library for computer-vision scientists and targeted for the multiple view geometry community. `http://openmvg.readthedocs.io/en/latest/openMVG/cameras/cameras/`. Accessed: 2017-17-07.

[60] Joseph O'rourke. *Art gallery theorems and algorithms*. Oxford University Press Oxford, 1987.

[61] Tolga Ozaslan, Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Inspection of penstocks and featureless tunnel-like environments using micro uavs. In *International Conference on Field and Service Robotics*, 2013.

[62] Fabio Pasqualetti, Joseph W Durham, and Francesco Bullo. Cooperative patrolling via weighted tours: Performance analysis and distributed algorithms. *Robotics, IEEE Transactions on*, 28(5):1181–1188, 2012.

[63] Picture credit for coordinate conversion. https://pixhawk.ethz.ch/software/coordinate_frame, 2017. [Online; accessed 14-July-2017].

[64] Pix4d: Photogrammetry software uses images to generate point clouds, digital surface, and terrain models. https://pix4d.com/. Accessed: July'19 2017.

[65] Quick summary of flight modes used by pixhawk autopilot. https://dev.px4.io/en/concept/flight_modes.html. Accessed: July'19 2017.

[66] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, number 3.2, page 5. Kobe, 2009.

[67] Robotics system toolboxâĎć provides algorithms and hardware connectivity for developing autonomous robotics applications for ground vehicles, manipulators, and humanoid robots. https://www.mathworks.com/products/robotics.html. Accessed: 2017-18-07.

[68] Ros wrapper for the swatbotics c++ port of the apriltag visual fiducial detector. https://github.com/personalrobotics/apriltags. Accessed: July'19 2017.

[69] Smart road facility managed by virginia tech transportation institute (vtti). http://www.apps.vtti.vt.edu/uav/. Accessed: July'19 2017.

[70] Ryan N Smith, Mac Schwager, Stephen L Smith, Burton H Jones, Daniela Rus, and Gaurav S Sukhatme. Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics*, 28(5):714–741, 2011.

[71] Stephen L Smith, Mac Schwager, and Daniela Rus. Persistent robotic tasks: Monitoring and sweeping in changing environments. *Robotics, IEEE Transactions on*, 28(2):410–426, 2012.

[72] Yoonchang Sung, Ashish Kumar Budhiraja, Ryan K Williams, and Pratap Tokekar. Distributed simultaneous action and target assignment for multi-robot multi-target tracking. *arXiv preprint arXiv:1706.02245*, 2017.

[73] Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys (CSUR)*, 45(2):24, 2013.

[74] tf is a package that lets the user keep track of multiple coordinate frames over time. http://wiki.ros.org/tf. Accessed: 2017-18-07.

[75] The intelÂő nuc is a powerful 4x4-inch mini pc. https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html. Accessed: July'19 2017.

[76] Pratap Tokekar, Ashish Kumar Budhiraja, and Vijay Kumar. Algorithms for visibility-based monitoring with robot teams. *arXiv preprint arXiv:1612.03246*, 2016.

[77] Pratap Tokekar, Volkan Isler, and Antonio Franchi. Multi-target visual tracking with aerial robots. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3067–3072. IEEE, 2014.

[78] Pratap Tokekar and Vijay Kumar. Visibility-based persistent monitoring with robot teams. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 3387–3394. IEEE, 2015.

[79] Tomlab: Optimization environment large-scale optimization in matlab. `http://tomopt.com/docs/quickguide/quickguide006.php`. Accessed: 2017-01-03.

[80] Tomlab/cplex handles large-scale mixed-integer quadratic programming (miqp) problems with linear and quadratic constraints. `http://tomopt.com/tomlab/products/cplex/`, 2017. [Online; accessed 14-July-2017].

[81] Use motion capture to fake gps for pixhawk platform. `https://dev.px4.io/en/advanced/fake_gps.html`. Accessed: 2017-18-07.

[82] Uses odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base. `http://wiki.ros.org/navigation`, 2017. [Online; accessed 13-July-2017].

[83] V.V. Vazirani. *Approximation algorithms*. Springer Publishing Company, Incorporated, 2001.

[84] Virginia unmanned systems article on use of uavs for infrastructure inspection. `http://vus.virginia.gov/air/infrastructure/`, 2017. [Online; accessed 18-July-2017].

[85] Pengpeng Wang, Ramesh Krishnamurti, and Kamal Gupta. Generalized watchman route problem with discrete view cost. *International Journal of Computational Geometry & Applications*, 20(02):119–146, 2010.

[86] Neal E Young. Sequential and parallel algorithms for mixed packing and covering. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 538–546. IEEE, 2001.

[87] Huili Yu, Kevin Meier, Matthew Argyle, and Randal W Beard. Cooperative path planning for target tracking in urban environments using unmanned air and ground vehicles. *IEEE/ASME Transactions on Mechatronics*, 20(2):541–552, 2015.

[88] Jingjin Yu, Sertac Karaman, and Daniela Rus. Persistent monitoring of events with stochastic arrivals at multiple stations. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[89] Jingjin Yu, Mac Schwager, and Daniela Rus. Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[90] Mengzhe Zhang and Sourabh Bhattacharya. Multi-agent visibility-based target tracking game. In *Distributed Autonomous Robotic Systems*, pages 271–284. Springer, 2016.

[91] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.