# Title of project placed here

Jack Moulson

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

07 Sept 2018

## Abstract

稳定的自主移动程序 是一个应用程序

Teams of robots often use task offloading to improve performance. Reliable Autonomous Mobile Programs (RAMPs) have recently been proposed for offloading within robot teams. To ensure optimal resource utilisation, the distribution of a workload should be fair across available resources. A key issue is how well RAMPs scale, i.e. how they perform as the number of RAMPs (tasks) and the number of robots increases.

最后扩展到160程个程序在12个robots

This project firstly evaluates the existing RAMP implementation by reproducing the existing RAMP implementation results on a maximum of 30 RAMPs on5 loaded and unloaded robot configurations.The project then scales to a maximum of 160 RAMPs and 12 robots, and identified several weaknesses with the existing implementation; as the number of robots increase, the number of non-optimal distributions also increase, and as the number of RAMPs increase the number of non-optimal distributions also increase. RAMP count has a more significant effect on the outcome. The project addresses these weakness by extending RAMPs to include the first ever implementation of competitive negotiating autonomous mobile programs. The project then evaluates the new implementation by increasing the number of cN-RAMPs to a maximum of 160 cN-RAMPs and available resources to a maximum of 12 robots. The project finds that the new implementation achieves optimal distributions more frequently across both dimensions of scale than the existing implementation.

## Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: ____________________ Signature: ______________________

## Acknowledgements

# Contents

# Chapter 1

# Introduction

## 1.1 Context

Teams of robots often comprise many heterogeneous compute resources, and may be required to complete a variety of task with differing compute needs. Task offload allows robots to share resources to better execute the workload. When tasks are offloaded, it is important to ensure that the distribution of tasks is fair (or optimal) across available resources, to guarantee performance and good resource utilisation. Autonomous Mobile Programs (AMP) [6] work to improve task offload distribution in teams of robots by allowing programs to autonomously choose where to execute based on their compute needs and available resources. Several extensions of AMPs exist, including Reliable Autonomous Mobile Programs (RAMP) [21], which aims to improve the fault tolerance of AMPs. Whilst the RAMP approach has been shown to have good performance on small scale systems, it is important to evaluate how well it works at increasingly scaled systems.

## 1.2 Contributions

The project firstly reproduces existing work on RAMPs [21] and identifies several weaknesses when the system is scaled in multiple dimensions. The project aims to address these issues with an extension of the RAMP model using cNAMPs. Finally this new approach is evaluated against the original RAMP approach. Specifically the contributions of the project are as follows:

- Reproduces previous task offload results [21] up to a maximum of 30 RAMPs on 5 loaded and unloaded robots with resulting distributions differing by at most 6% across 18 executions on 6 different robot configurations (Section 3.1).
- Scales beyond previous experiments up to a maximum of 160 RAMPs and 12 robots, by increasing the number of RAMPs, the number of robots, and both simultaneously. We find that RAMP count is a more likely to cause non-optimal distribution performance (Section 3.2).
- Identifies several weaknesses of the existing RAMP implementation at these scales:

  - We find that as the number of robots increase, the frequency of non-optimal distributions increases by 44% across 15 executions on 5 different robot configurations (Section 3.2.2),
  - We find that as the number of RAMPs increase, the frequency of non-optimal distributions goes from 0 to 100% across 18 executions on 6 different robot configurations (Section 3.2.1).

- Addresses these distribution weaknesses by extending the RAMP implementation to include the first ever implementation of competitive Negotiating AMPs (cNAMP) (Section 4.1).
- Evaluates Negotiating Reliable Autonomous Mobile Programs (cN-RAMP) implementation on scaling the number of cN-RAMPs up to a maximum of 160 and the number of robots up to a maximum of 12:
  - We find that cN-RAMPs achieve optimal distributions more frequently than RAMPs, and the number of steps required to reach optimal distributions in near and non optimal distributions are lower than those achieved by RAMPs (Section 4.2).
  - We find that as the number of robots increase, non-optimal distributions are achieved between 36-66% less frequently than by the RAMP approach across 15 executions on 5 different robot configurations (Section 4.3).

# Chapter 2

# Background

## 2.1 Multi-Robot Systems

Given the rate of technological advancements over the past decade or so, the adoption of robotics has become increasingly widespread. In most domains, the performance of multiple robots outperforms a single robot, both with respect to cost, efficacy and domain potential [9][1]. This has led to multi-robot systems becoming a key area of research within the field of robotics in general [20].

Multi-robot systems are those that comprise several autonomous and intelligent robots, and are often generally categories as either *cooperative* or *collective*, and in the case of cooperative systems, further categorised based on communication architecture [15]. The applications of multi-robot systems are widespread and varied. Examples of multi-robot uses include the deployment of robot teams in urban environments to provide situational sensing and awareness [4], and toxic chemical plume tracing [17]. The introduction of more than a single robot in a system, however, does give rise to the need for intra-robot communication [7], and thus concerns of scalability, reliability, and task management.

### 2.1.1 Scalability

In multi-robot systems where the computational capacity (as in the case of heterogeneous systems) and/or number of robots is variable, the question of scalability and how it affects performance becomes important [10]. Specifically, the ability of the system to scale to deal with increased work load [2]. With respect to both computational and physical variability, scalability can generally be split into vertical and horizontal scalability [12]. Vertical scalability is the increase in computational capacity of a single system node, whereas horizontal scalability is the increase in the number of nodes available.

The performance of these two types of scalability is dependent on the types of computation being executed by a system. Where code executions computationally mild and are thus unlikely to saturate a node's resources, the problem lends itself to vertical scaling [Isaac Jordan]. Where code executions are more computationally intensive, execution offloading becomes a more realistic approach, and thus scaling horizontally. Further, Michael et al. [12] suggest that for scenarios where workloads are *highly parallel*, horizontal scalability greatly out-performs vertical scalability.

### 2.1.2 Reliability

As with scalability, the introduction of multiple autonomous robots introduces concerns of reliability, or fault tolerance. Whilst fault tolerance is still a concern with a single robot, it is emphasised where communication and task offload become relevant. In a context of task offload, fault tolerance is especially important in ensuring tasks are completed and results are not lost.

Faults within a multi-robot systems can occur as a result of several types of failure, and can generally be defined as *the inability of the robot or the equipment used with the robot to function normally* [3]. In the context of the systems discussed here, failures can be broken down into robotic (hardware) failure and communication failure [21]. Hardware failure may be a sensor or actuator failing due to loss of power and thus losing the current state of execution of some process without the ability to resume. Network failure may refer to the loss of connection between robots and thus being unable to successfully offload work or return results.

### 2.1.3 Load Distribution

Load distribution comes from the idea that within systems of robots, the resource usage for each robot may differ, and that by distributing tasks from *loaded* robots to *unloaded* robots, that the processing of the system as a whole may be improved. Assuming a system of robots that can communicate and are able to distribute tasks to optimise process execution, the method of load distribution (or task allocation) across the available robots is important to consider. Broadly, the concerns of task allocation within a multi-robot system are to decide *when* to offload from one robot to another, and *where* the task should be offloaded to, depending on the current state of the system. Both these should be taken into account when determining if an offload should take place.

**When** Determining when an offload will benefit a system and should take place requires each robot to have some knowledge of the current state of the system. Having this information allows robots to decide when an offload would be beneficial compared to continuing to execute a task locally. Offloading a task has added cost in the form of communication of the task itself. In a scenario where the speed of communication across the network is great enough that the offloading becomes less beneficial. Generally, the decision whether to offload a task takes into account computation time and communication time based on the following inequality:

$$Comp_x > Comp_y + Comm_{xy}$$

where $Comp_x$ is the time to execute a task on robot $x$, and $Comm_{xy}$ is the time to send a task from robot $x$ to robot $y$. If a robot deems the inequality to hold for at least one other robot within the system, then the task should be offloaded.

**Where** The decision as to where to offload a task to follows on fairly simply from the decision when to offload. Given that the inequality holds for at least one other robot, in the case where only a single other robot satisfies the inequality then the decision is trivial. In the case where multiple robots satisfy it, the simple and (possibly) naive solution is to offload to the robot where RHS of the inequality is minimised. Of course this naive strategy may not always achieve optimal offloads, and further negotiation strategies are discusses in subsequent sections.

## 2.2 Robot Operating Software (ROS)

In most multi-robot systems, there exists some form of *robotic middle-ware*. These middle-wares comprise software libraries and technologies that aid in the implementation and development of multi-robot systems, by helping to manage the heterogeneity, communication, and general system complexity [8]. ROS is one such robotic middleware that is widely used and accepted as a standard for multi-robot system development [16].

ROS provides a range of features to help facilitate the development of multi robot systems including but not limited to hardware abstraction, low-level device control, and inter-device/process message-passing [11]. Since message passing is the method of implementation for inter-robot communication in the ROS ecosystem, discussions of ROS shall focus primarily on this.

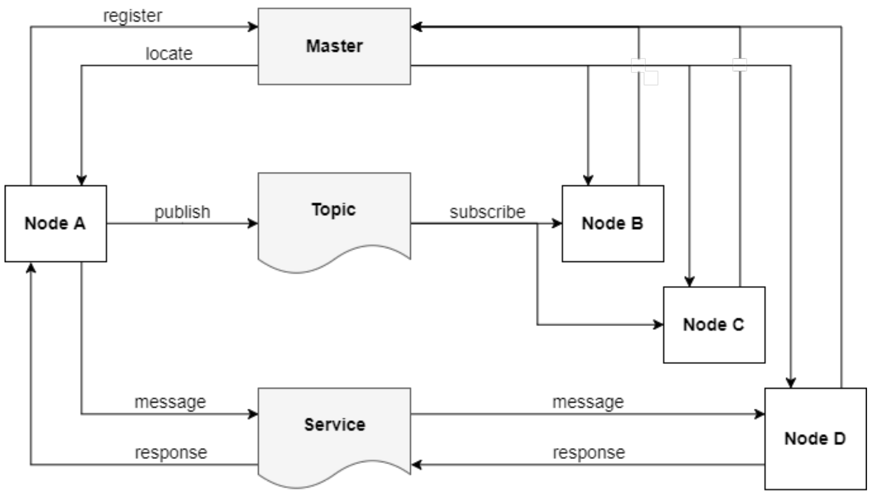

Figure 2.1: ROS Communication

ROS uses a graph-like architecture to organise processes and communication. Processes are known as nodes and are they atomic unit of computation within ROS. They can interact with other nodes using a variety of methods depending on the specific functional needs. ROS documentation [14] states that nodes should be fine-grained, providing a high separation of functional concerns. This provides limited node failure fault tolerance, and reduced overall code complexity. Nodes are aided by a master node that allows nodes location other nodes and topics. Once communication between nodes has been established, the nodes can communicate freely without the use of the master, preserving the decentralised nature of the architecture.

Communication is provided through two main methods; a publisher/subscriber method using *topics*, and a more direct request-response method using *services*. Topics allow nodes to send and receive data by publishing strongly typed messages to a topic, to which nodes can then subscribe and receive the messages. Fig 2.1 shows an example of a simple ROS system, comprising four nodes. Services provide the ability for a sender to receive a returned value, similar to an RPC invocation.

All communication in ROS is strongly types, and all message types are defined in advance.

## 2.3 Reliable Autonomous Mobile Programs (RAMPS)

RAMPS, which are themselves an extension of AMPS [6], take further the idea of task allocation by considering tasks as mobile programs that are aware of their resource needs and can dynamically and autonomously allocate themselves. RAMPS as proposed in [21], build upon the core AMP to provide fault tolerance for failures such as robot failure or communication failure. RAMPs are shown to have good performance in load distribution for both unloaded and loaded systems and successfully deal with several failure conditions.

RAMPs/AMPs facilitate this decentralised load balancing by having an awareness of their resource needs based on the required task, and can make informed decisions about where to execute by querying each location from the set of possible locations. AMPs essentially represent modular autonomous task execution “packages”. In the context of multi-robot systems then locations are the robots themselves that have limited hardware and software constraints. Importantly, RAMPs benefit from full autonomy, meaning they can and should perform logically and functionally independently. This allows RAMPs to take advantage of a decentralised and fully distributed approach, improving overall reliability, with the small cost of ensuring system consistency.

Based only on local resource RAMP needs and remote location statistics, however, RAMPs are prone to forming non-optimal distributions across possible execution locations, due to greedy location selection, known as thrashing [5][Ivan]. This effect appears to be more prevalent as the scale of a multi-robot system increases [Ivan paper]. The effect of scaling to greater extents shall be explored further in the course of this project. RAMPs also slightly diverge from the true AMP implementation by making use of a single load server, reducing the decentralised nature of the system. However the autonomous nature of the RAMPs is preserved as much as possible given that the load balancer acts as a reporter/recorded only, with no actual decision making functionality.

To alleviate this trashing phenomenon, AMP level negotiation processes can be put in place to help prevent the choice of a location that would result in a non-optimal distribution. One such negotiation strategy directly related to AMPs is known as cNAMPs [5]. cNAMPs introduce the concept of a “request representative”, which are sent by AMPs to locations prior to a choice being made to ensure that the resources of the intended location are still suitable for the task execution. Based on this each AMP can then make a more informed decision. cNAMPs are shown to reduce the redundant moves required (i.e. movements required from a non-optimal distribution to an optimal one) by achieving an optimal distribution in the first case. A combination of cNAMP and RAMP models may outperform the existing RAMP approach in terms of achieving optimal load distribution at scale already tested for RAMPs and at greater scales also.

# Chapter 3

# Experiments and Results

To evaluate the RAMP model, the performance of the RAMP implementation was first replicated based on the experiments performed in [21]. Experiments were run on both unloaded and loaded systems. The results were then used to inform the subsequent experiments based on an analysis of the results and the RAMP model in general. These experiments consisted of exploring two dimensions of scale; increasing the number of robots, and increasing the number of RAMPs, as well as a combination of the two. The aim was to evaluate 1) how well RAMPs performed at achieving optimal or near-optimal distributions as the number of possible locations increased, 2) how well they performed as the number of RAMPS increased relative to the number of locations, and 3) under what conditions, if any, optimal distributions were more consistently not achieved.

To ensure as similar an execution environment as possible to the original experiments run, all tests were performed using a collection of identical Raspberry Pi 3 Model Bs. Six of these were part of kit-car set-ups [18]. The remaining Raspberry Pi were stand along. Each had the following specifications:

- Quad Core, 1.2GHz, 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN
- Ubuntu 16.04 w/ ROS

Each RAMP performed an identical route planning task implemented and solved using MiniZinc [13] and Gecode [19]. This was the task performed in the initial experiments in [21]. A single task was solved in approximately 50s and executed using a single core.

## 3.1 RAMP Replication and Analysis

As a performance baseline, the experiments reported in [21] were reproduced. A team of 5 robots was used, and the number of RAMPS was varied initially from 5 to 30. The distribution was recorded by observing the number of RAMPs that executed on each possible location. Each set up was run 3 times and the number of optimal, near-optimal, and non-optimal distributions were recorded. In this case, **optimal** refers to distributions where no RAMP movements (from one robot to another) could be made to improve the overall distribution. **Near-optimal** refers to distributions where the movement of a single RAMP would create an optimal one, and accordingly, **non-optimal** distributions are those that require movement of two or more RAMPs to achieve optimality.

| | Robots | | | | |
|---|---|---|---|---|---|
| # of RAMPs | R1 | R2 | R3 | R4 | R5 |
| 5 | 1 | 1 | 1 | 1 | 1 |
| 10 | 1 | 2 | 2 | 2 | 3 |
| 15 | 2 | 4 | 3 | 3 | 3 |
| 20 | 4 | 4 | 4 | 4 | 4 |
| 25 | 4 | 5 | 6 | 5 | 5 |
| 30 | 8 | 6 | 6 | 5 | 5 |

(a) Replicated Distributions

| | Robots | | | | |
|---|---|---|---|---|---|
| # of RAMPs | R1 | R2 | R3 | R4 | R5 |
| 5 | 1 | 1 | 1 | 1 | 1 |
| 10 | 1 | 2 | 3 | 2 | 2 |
| 15 | 2 | 3 | 3 | 3 | 4 |
| 20 | 3 | 4 | 4 | 5 | 4 |
| 25 | 4 | 6 | 5 | 5 | 5 |
| 30 | 7 | 5 | 6 | 6 | 6 |

(b) Distributions reported in [21]

Table 3.1: Unloaded RAMP Distributions

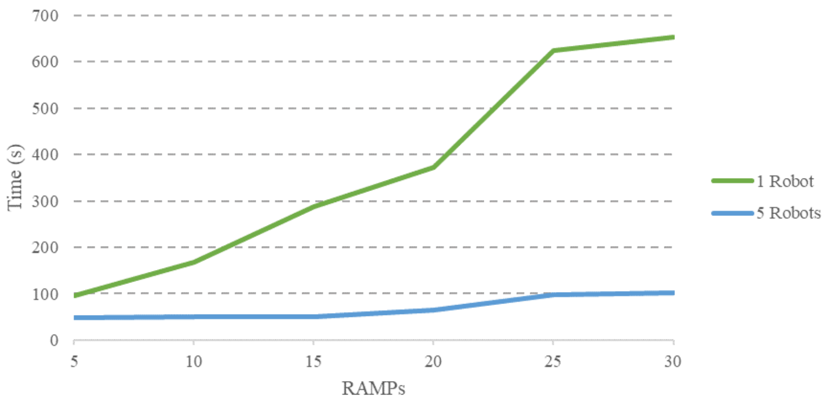

Figure 3.1: Unloaded Execution Times

The distributions observed at each number of RAMPs are shown in Tables 3.1. The data shown is a representative set of one run. The replicated results in Table 3.1a show that in all almost all cases the RAMPs achieved optimal or near-optimal distributions, with higher numbers of RAMPs forming non-optimal distributions. Over all runs, 44% of distributions were optimal, 39% were near-optimal, and 17% were non-optimal compared to 46%, 43% and 11% respectively in [21]. This shows a difference of at most 6% across each distribution frequency. Although explored further in subsequent sections, it is interesting to note here that higher number of RAMPs appeared to result in non-optimal distributions *more frequently* than lower numbers of RAMPs. Fig 3.1 also shows the execution time for equal numbers of RAMPs executed on one and five robots. Given that RAMPS attempt to distribute evenly across possible locations, it follows that in larger teams of robots, more of the available robot cores will be used in parallel and thus reducing the overall execution time. The results strongly suggest this to be the case.

Experiments with certain robots being placed under load prior to the initial distribution performed in [21] were also replicated. The aim was to observe the reaction of the RAMPs to robots with differing available resources. Using a system stress tool, the CPU usage for each loaded robot was artificially raised to 100% before the RAMPs were run. Table 3.2 shows robots 1, 2, 3, and 4

| | Robots | | | | |
|---|---|---|---|---|---|
| RAMPs | R1 (L) | R2 (L) | R3 (L) | R4 (L) | R5 (U) |
| 5 | 0 | 0 | 1 | 1 | 3 |
| 10 | 1 | 1 | 1 | 2 | 5 |
| 15 | 2 | 2 | 2 | 2 | 7 |
| 20 | 2 | 2 | 2 | 3 | 8 |
| 25 | 4 | 6 | 3 | 3 | 9 |
| 30 | 5 | 5 | 7 | 4 | 9 |

(a) Replicated Distributions

| | Robots | | | | |
|---|---|---|---|---|---|
| RAMPs | R1 (L) | R2 (L) | R3 (L) | R4 (L) | R5 (U) |
| 5 | 0 | 0.33 | 0.33 | 0.67 | 3.67 |
| 10 | 0.67 | 1 | 0.67 | 1 | 6.67 |
| 15 | 2 | 2 | 2 | 2.33 | 6.67 |
| 20 | 3 | 3 | 3 | 2.33 | 8.67 |
| 25 | 4.33 | 3.67 | 4.33 | 4 | 8.67 |
| 30 | 5.67 | 5.33 | 5.67 | 5 | 8.33 |

(b) Distributions reported in [21]

Table 3.2: Loaded RAMP Distributions

under load, and robot 5 unloaded. In all cases the the results show that the RAMPs highly favoured robots with more available resources over ones that were under greater load. This is in line with the original expectations. The relationship between the number of RAMPs and optimal vs. non-optimal distributions across specifically loaded robots appeared less acute than in the system of unloaded robots. This is perhaps due to loss of information at maximum loads, discussed further in subsequent sections.

The results obtained in the repeated unloaded and loaded experiments are very similar to the original results. Comparing Table 3.1a and 3.1b shows that 4 rows match, and comparing 3.2a and 3.2b shows that 3 rows match (where every cell has a difference of at most 1). The RAMP model successfully achieved a high percentage of optimal and near-optimal distributions across possible execution locations and successfully reduced the overall completion time by around 84%. The RAMP model also has proven to correctly favour execution locations with greater available computational resources where possible. However whilst most of the observed distributions were optimal or near-optimal, a notable number remained non-optimal at certain numbers of RAMPs.

## 3.2 Scaling and Optimality

The number of RAMPs and robots were increased to explore the relationship between these two variables and the frequency of non-optimal distributions. The expectation was that increasing either would lead to an increase in the number of non-optimal distributions.

### 3.2.1 Scaling the number of RAMPs

The number of RAMPs was increased from the original range of 5-30 with steps of 5, to a range of 5-160, doubling the number of RAMPs in each step to give a greater coverage. This was performed on a team of 5 unloaded robots. Again, each was run a total of three times.

Table 3.3 shows the distributions of RAMPs across robots at each of these increased levels. The results show non-optimal distributions being achieved at higher number of RAMPs. As the number of RAMPs increases, the number of location evaluations being made increases following the general pattern $x \times y$ where $x$ is the number of robots and $y$ is the number of RAMPs to execute. At the highest level presented here, this results in 800 location evaluations taking place. It follows that the greater number of location evaluations taking place, the greater the volume of incorrect decisions made, and thus an increase in the frequency of non-optimal distributions. With the greater number of evaluations taking place, the likelihood of scheduling conflicts also increase, especially since no realistic negotiation strategy exists.

| | Robots | | | | |
|---|---|---|---|---|---|
| # of RAMPs | R1 | R2 | R3 | R4 | R5 |
| 5 | 1 | 1 | 1 | 1 | 1 |
| 10 | 1 | 2 | 2 | 2 | 3 |
| 20 | 4 | 4 | 3 | 4 | 5 |
| 40 | 8 | 8 | 11 | 7 | 6 |
| 80 | 20 | 16 | 14 | 15 | 15 |
| 160 | 28 | 29 | 34 | 32 | 37 |

Table 3.3: Distributions for Increasing Numbers of RAMPs

Fig 3.2 shows the frequency at which non-optimal distributions were observed across the three runs for each number of RAMPs. The observed frequencies show that above a certain level of RAMPs, they struggle to achieve optimal or near-optimal distributions in any case. Important to not here that the "severity" of the non-optimal distributions are not recorded, and thus there is no scale of how badly a distribution might be compared to another. However, the results do reinforce the idea that as the number of RAMPs increase, the frequency at which non-optimal distributions are observed increases. This may be a result of the increased number of possible decisions being made, and thus likelihood of making a non-optimal choice. The number of non-optimal distributions may also result from a lack of appropriate information exchange strategies between RAMPs. The effects of scaling both the number of RAMPS and robots may exacerbate this by increasing the likelihood of scheduling conflicts occurring between RAMPs.

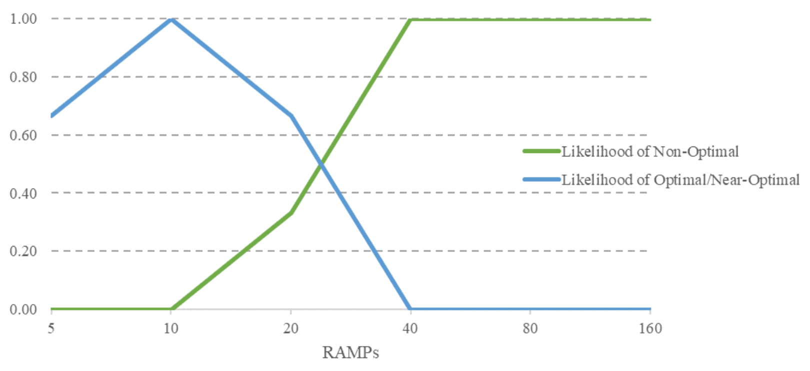

Figure 3.2: Likelihood of Distributions for Increasing Numbers of RAMPs on Five Robots

### 3.2.2 Scaling the number of Robots

It was also important to increase the number of robots to observe the affect of providing a greater number of possible execution locations for the RAMPs. For each number of robots, the number of RAMPs was set at levels that were multiples of the number of robots. This allowed for each number of robots to have exact optimal-distributions to aim for. As with previous sections the important observations were the number of optimal and non-optimal distributions. Each number of robots was run three times, with varying levels of RAMPS between $x$ and $6x$ where $x$ is the number of robots, and the distribution counts recorded.

|  | Distributions |  |  |
|---|---|---|---|
| # of Robots | Optimal | Near-Optimal | Non-Optimal |
| 5 | 8 | 7 | 3 |
| 6 | 9 | 6 | 3 |
| 8 | 8 | 6 | 5 |
| 10 | 3 | 3 | 9 |
| 12 | 3 | 4 | 11 |

Table 3.4: Distributions for Increasing Numbers of Robots

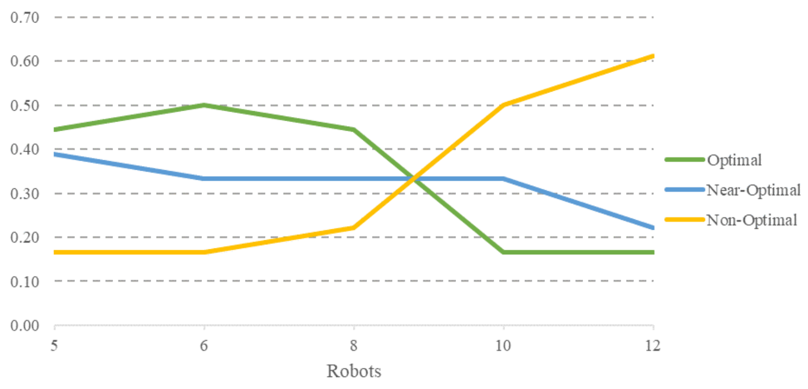

Figure 3.3: Likelihood of Distributions for Increasing Numbers Robots

Table 3.4 and Fig 3.3 both show that as the number of robot locations increase, the number of optimal and near-optimal distributions decrease. The reason for this is likely similar to that of the observed effect from increasing the number of RAMPs; the more decisions that are required to take place, the greater the chance that a RAMP may decide wrongly. The number of optimal distributions does not drop to zero completely though, showing that RAMPs are still able to achieve optimal distributions, especially when the number of RAMPs is low. It seems that an increase in the number of RAMPs is more likely to result in non-optimal distributions than an increase in the number of robots.

### 3.2.3 Scaling RAMPs and Robots

Finally, both RAMP count and robot count were increased simultaneously. In this case the RAMP count was keep constant across each increasing number of robots contrastingly with previous experiments. Each instance was run three times and for each the number of non-optimal distributions was recorded.

Results are plotted as a surface chart in Fig 3.4. As expected, the frequency of non-optimal distributions remained low across all robot counts for low numbers of RAMPs, with the frequency increasing towards higher number of RAMPs. Generally, the frequency of non-optimal distributions appear higher with lower numbers of robots for all RAMP counts, suggesting that RAMPs struggle to optimally distribute when the available space is lower. Where large numbers of RAMPs are competing for the same locations, it follows that the number of scheduling conflicts may increase as a result of many RAMPs allocating the same locations, leading to a RAMP/location bottleneck.

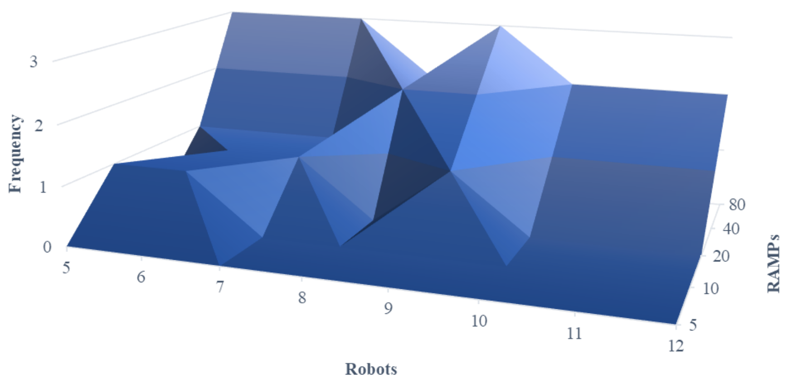

Figure 3.4: RAMPs and Robots

This suggests that, whilst sheer number of evaluations does lead to greater number of non-optimal distributions, when many RAMPs “clash” for the same few locations, this has a greater impact overall. This highlights the need for some information exchange between RAMPs, possibly through a negotiation scheme such as cNAMPs.

# Chapter 4

# Implementation and Evaluation of Negotiating RAMPs (cN-RAMP)

Negotiating AMPS, cNAMPs, were proposed to minimise non-optimal distributions by communicating *intended* moves between AMPs [5]. This allows AMPs to signal that they are moving to a location without having to wait to transfer before the information becomes available to other AMPs. In this chapter we extend the RAMP implementation with negotiation to produce cN-RAMPs (Section 4.1). The new implementation was then evaluated against increasing numbers of RAMPs, and increasing numbers of robots. The aim of this section was to evaluate 1) how well the this approach improves at achieving optimal or near-optimal distributions as the number of RAMPs was increased in relation to the number of locations (Section 4.2), and 2) how well it improved as the number of locations is increased (Section 4.3).

## 4.1 Implementation

The key functional elements of the cNAMP implementation are the request agent, and the load server load values. When a cNAMP decides on a location to move to, a request agent is first sent to the corresponding load server (in the current RAMP implementation, a single load server acts for all locations, where in a more practical implementation each location would use an independent load server). The request confirms whether the load server is still a suitable location to move to, and if so, confirms this with the load server and the cNAMP. The load server then updates the relevant loads, and the cNAMP begins moving.

The load server itself maintains two load values; actual and committed load. Actual load represents the current real CPU usage of the locations, which is updated continuously. Committed load presents a combination of the actual load on the location as well as any load from cNAMPs that have "reserved" space by means of a request. The committed load is thus always $\geq$ actual load. Committed load is updated with actual load and also when a request is approved.

**Request Agent** The request agent uses a ROS Service to interact with the load server, allowing request/response interaction. Once a location has been chosen, a service call is made which returns the required data for the request agent to ensure the location is suitable. The request agent confirms the location against other locations to ensure the execution time is still minimised. The request agent then sends a final message to the load server confirming or rejecting the location. In the event that a location is rejected, the cN-RAMP continues to inspect locations. Fig 4.1 shows a logical flow of cN-RAMP and Load Server interaction. It is important that the request functionality is kept on the cN-RAMP as opposed to being performed on the load server itself in the interest of preserving as much of the autonomy of the cN-RAMP as possible.

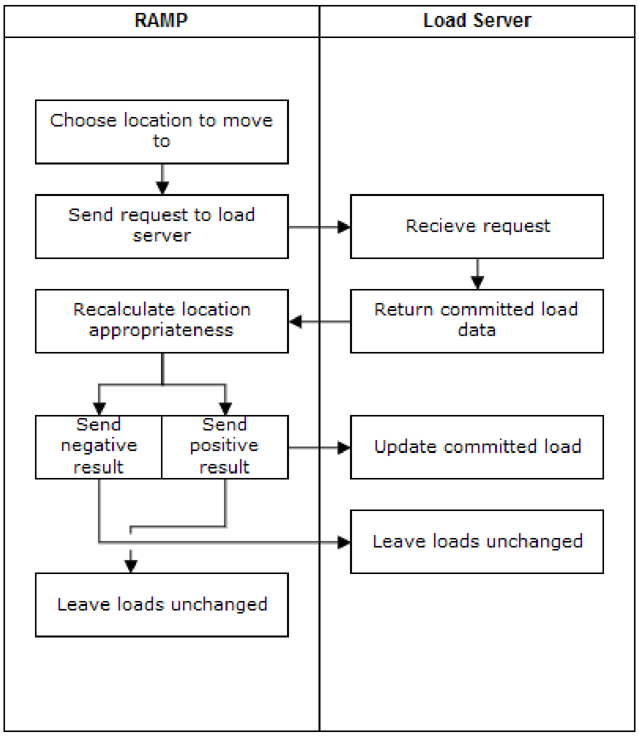

Figure 4.1: cNAMP Request Agent Functional Process

**Actual and Committed Load** Actual load is updated from each location every 0.5 seconds. CPU usage is reported by means of a ROS Topic, which locations publish to, and the load server subscribes from. This is common with the RAMP implementation. The committed load stores a value on the load server that represents the number of cN-RAMPs that have requested a move to that location. When the committed load is requested, the actual and committed load are combined to give a full committed load for the location.

## 4.2 Scaling the number of RAMPs

The new cN-RAMP implementation was evaluated on increasing numbers of cN-RAMPs by increasing the cN-RAMP count from 5-160, doubling the number each step, on a total of five unloaded robots. Each experiment was run three times.

Table 4.1 shows the resulting distributions for each cN-RAMP count. Although not achieving optimal distributions in all instances, the results suggest that the cN-RAMP approach achieves

|  | Robots |  |  |  |  |
|---|---|---|---|---|---|
| # of RAMPs | R1 | R2 | R3 | R4 | R5 |
| 5 | 1 | 1 | 1 | 1 | 1 |
| 10 | 2 | 2 | 2 | 2 | 2 |
| 20 | 4 | 4 | 4 | 4 | 4 |
| 40 | 8 | 8 | 8 | 9 | 7 |
| 80 | 18 | 16 | 16 | 15 | 15 |
| 160 | 31 | 30 | 35 | 32 | 32 |

Table 4.1: Distributions for Increasing Numbers of cN-RAMP

distributions that are closer to optimal than those achieved with the RAMP approach. Comparing Table 4.1 and 3.3 shows that the distributions at higher cN-RAMP/RAMP counts are much closer to an optimal distribution under the cN-RAMP approach than the RAMP approach, requiring three and seven moves at 160 to reach an optimal distribution respectively. The frequency of non-optimal distributions is also lower as can be seen by comparing Fig 4.2 and Fig 3.2. As with the RAMP increase, the greater the number of cN-RAMPs, the more likely that a non-optimal distribution will be achieved, however cN-RAMPs achieve non-optimal distributions around 67% less frequently than RAMPs. This is calculated by comparing the likelihoods for both implementations across all 18 executions on all 6 robot configurations.

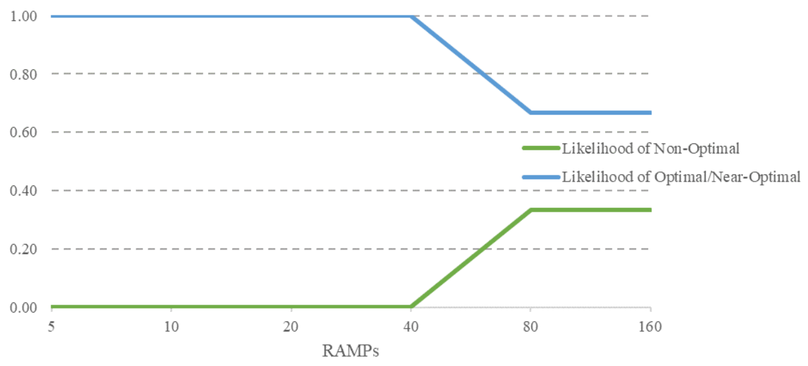

Figure 4.2: Likelihood of Distributions for Increasing Numbers of cN-RAMP on Five Robots

Whilst the non-optimal distributions are not eliminated, *measurements of distribution frequency across 18 executions on 6 robot configurations show that cN-RAMPs achieve optimal distributions more frequently than RAMPs, and the number of steps required to reach optimal distributions in near and non optimal distributions are lower than those achieved by RAMPs.*

## 4.3 Scaling the number of Robots

The cN-RAMPs were also evaluated against increasing numbers of locations, by increasing the number of available robots. As with the experiments in Section 3.2.2, the number of robots was increased from 5 to 12, with numbers of cN-RAMPs corresponding to the number of robots. Each instance was repeated three times.

The results show that the cN-RAMPs approach appears to achieve optimal distributions more fre-

|  | Distributions |  |  |
|---|---|---|---|
| # of Robots | Optimal | Near-Optimal | Non-Optimal |
| 5 | 13 | 4 | 1 |
| 6 | 12 | 4 | 2 |
| 8 | 9 | 4 | 5 |
| 10 | 8 | 4 | 6 |
| 12 | 6 | 5 | 7 |

Table 4.2: Distributions for Increasing Numbers of Robots

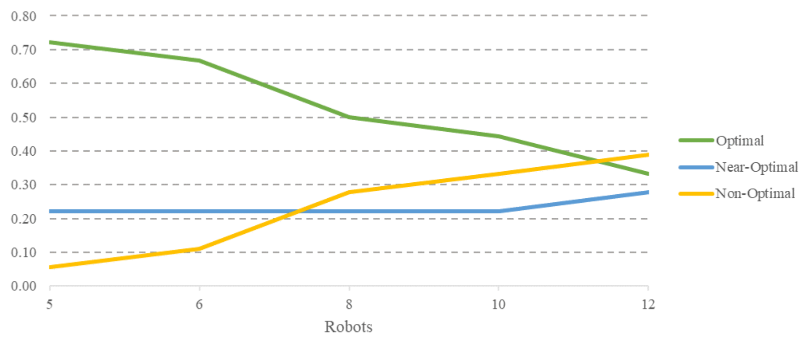

Figure 4.3: Likelihood of Distributions for Increasing Numbers of Robots

quently than RAMPs, at all numbers of robots. Comparing Table 4.2 and Table 3.4 shows that in a single instance at five robots, the cN-RAMPs achieve 62.5% more optimal distributions than RAMPs. At twelve robots this increases to around 100% increase. Improvements are lower for fewer robots however this is possibly in part due to the already reasonable performance by RAMPs at lower numbers of robots.

Further, comparing Fig 4.3 and Fig 3.3 shows that overall, optimal and near-optimal distributions are achieved around 10% more frequently respectively at twelve robots, and non-optimal distributions are achieved around 20% less frequently than by RAMPs. *Measurements of distribution frequency across 15 executions on 5 robot configurations show that cN-RAMPs achieve optimal distributions more frequently, at all number of robots presented. The results from both sets of experiments reinforce the idea that appropriate information exchange between AMPs allows improved coordination for task allocation, resulting in better distributions of work over available resources.*

# Chapter 5

# Conclusion

## 5.1 Summary

This project firstly evaluates the existing RAMP implementation by reproducing the existing RAMP implementation results on a maximum of 30 RAMPs on5 loaded and unloaded robot configurations (Section 3.1). The results show a difference from the initial experiments of at most 6% across all 18 executions on 6 robot configurations. The project then scales to a maximum of 160 RAMPs and 12 robots (Section 3.2), and identified several weaknesses with the existing implementation; as the number of robots increase, the number of non-optimal distributions also increase by around 100% seen in Table 3.4, and as the number of RAMPs increase the number of non-optimal distributions also increase by around 44% seen in Fig 3.2. RAMP count has a more significant effect on the outcome.

The project addresses these weakness by extending RAMPs to include the first ever implementation of competitive negotiating autonomous mobile programs (Section 4.1). The project then evaluates the new implementation by increasing the number of cN-RAMPs to a maximum of 160 cN-RAMPs (Section 4.2) and available resources to a maximum of 12 robots (Section 4.3). The project finds that at increasing number of CN-RAMPs the new implementation achieves non-optimal distributions around 67% less frequently than RAMPs as seen in Fig 4.2. The cN-RAMP implementation also achieves non-optimal distributions between 36-66% less frequently over increasing numbers of robots, as seen in Fig 4.3.

## 5.2 Future Work

Whilst the cN-RAMP implementation is successful in achieving more optimal and near optimal distributions, it is implemented using a single load server. This reduces the distributed nature and autonomy of the AMPs. In more practical implementations, each robots should have an independent load server, which raises questions of information consistency across the system. Dealing with multiple load servers whilst sharing information is a difficult task and may be explored in future extension of cN-RAMPs.

Also not fully explored in this project is the effect of loaded systems on cNAMP load distribution.

The distribution of tasks across loaded robots is briefly evaluated and suggests a lack of appropriate decision making when locations are already fully loaded. This is likely due to comparative information loss when multiple robots are at 100% usage. Future work may explore a more comprehensive strategy for choosing a location when this is the case, which may include tracking multiple loads such as AMP load and existing load.

Appendix A

# First appendix

## A.1 Section of first appendix

Appendix B

# Second appendix

# Bibliography

[1] Tamio Arai, Enrico Pagello, and Lynne E. Parker. Guest editorial advances in multirobot systems. In *IEEE Transactions on Robotics and Automation*, volume 18, pages 655–661, Oct 2002.

[2] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the Second International Workshop on Software and Performance - WOSP '00*, pages 195–203, New York, New York, USA, 2000. ACM Press.

[3] J. Carlson, R.R. Murphy, and A. Nelson. Follow-up analysis of mobile robot failures. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, pages 4987–4994 Vol.5. IEEE, 2004.

[4] L. Chaimowicz, A. Cowley, D. Gomez-Ibanez, B. Grocholsky, M. A. Hsieh, H. Hsu, J. F. Keller, V. Kumar, R. Swaminathan, and C. J. Taylor. Deploying Air-Ground Multi-Robot Teams in Urban Environments. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 223–234. Springer-Verlag, Berlin/Heidelberg, 2005.

[5] Natalia Chechina, Peter King, and Phil Trinder. Using Negotiation to Reduce Redundant Autonomous Mobile Program Movements. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 343–346. IEEE, aug 2010.

[6] Xiao Yan Deng, Phil Trinder, and Greg Michaelson. Autonomous Mobile Programs. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 177–186. IEEE, 2006.

[7] Rajesh Doriya, Siddharth Mishra, and Swati Gupta. A brief survey and analysis of multi-robot communication and coordination. In *International Conference on Computing, Communication & Automation*, pages 1014–1021. IEEE, May 2015.

[8] Ayssam Elkady and Tarek Sobh. Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. *Journal of Robotics*, 2012:1–15, May 2012.

[9] Avinash Gautam and Sudeept Mohan. A review of research in multi-robot systems. In *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, pages 1–5. IEEE, Aug 2012.

[10] Eric Klavins. Communication Complexity of Multi-Robot Systems. *Algorithmic Foundations of Robotics V*, 7(December):275–292, 2004.

[11] Aaron Martinez and Enrique Fernández. *Learning ROS for Robotics Programming*. 2013.

[12] Maged Michael, José E. Moreira, Doron Shiloach, and Robert W. Wisniewski. Scale-up x scale-out: A case study using nutch/Lucene. In *Proceedings - 21st International Parallel and Distributed Processing Symposium, IPDPS 2007; Abstracts and CD-ROM*, pages 1–8. IEEE, 2007.

[13] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[14] Open Source Robotics Foundation. Nodes - ROS Documentation. `http://wiki.ros.org/Nodes`, 2012. Accessed: 2018-07-13.

[15] Lynne E. Parker. Multiple Mobile Robot Systems. In *Springer Handbook of Robotics*, pages 921–941. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[16] Morgan Quigley, Ken Conley, Brian P Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System, Jul 2009.

[17] Diana Spears, Dimitri Zarzhitsky, and David Thayer. Multi-robot chemical plume tracing. *Multi-Robot Systems. From Swarms to . . .* , III:211–221, 2005.

[18] SunFounder. Smart Video Car Kit for Raspberry Pi. `https://www.sunfounder.com/rpi-car.html`, 2018. Accessed: 2018-08-27.

[19] Gecode Team. Gecode - An open, free, efficient constraint solving toolkit. `http://www.gecode.org/`, 2018. Accessed: 2018-08-21.

[20] Pedro U. Lima and Luis M. Custódio. Multi-Robot Systems. In *Innovations in Robot Mobility and Control*, pages 1–64. Springer, Berlin, Heidelberg, Aug 2005.

[21] Valkov, Ivan. RAMPs: Reliable Autonomous Mobile Programs for Teams of Robots. Master's thesis, University of Glasgow, 2018.