Investigation into Swarm-based Cooperative Behaviour in Execution of Open Field Agricultural Tasks

Alireza Janani

A doctoral project report submitted in partial fulfilment of the requirements of
Sheffield Hallam University
for the degree of Doctor of Professional Studies Doctor of Philosophy

Sheffield Hallam University | Materials and Engineering Research Institute

Sheffield Hallam Univeristy
Materials and Engineering Research Institute
UK
March 2018

Abstract

Because of the significant drop in the number of farmers and increase in the earth population, the use of autonomous farming units including unmanned tractors is becoming more and more popular. However, relying on a single autonomous farming unit to carry out the entire task on a large field is inefficient. Using multiple autonomous tractors bring more efficiency, however, without cooperation this attempt will fail (Mataric et al., 1995). This cooperation can be achieved by an appropriate task allocation and coordination mechanism between the participating units. The current trend in this field is to use direct forms of communication in any form of directional or broadcasting meaningful messages among the group. The messages assist the group to identify the state of the task, assigned workload, collision and congestion avoidance, and etc. These forms of approaches are fast and efficient when units are within the communicating signal range.

In this thesis, we aim to investigate the feasibility of cooperative execution of open field farming task including spraying and ploughing while inter-team interaction is other than direct communication methods. For every task, an algorithm is suggested and an appropriate mathematical model is presented. Then, using ROS Stage simulation environment, each algorithm is implemented and multiple tests are conducted. Finally, the simulation results and the correspondent mathematical results are compared and appropriate modifications are suggested.

Candidate's Statement

I declare that this work has been conducted in accordance with the regulations of Sheffield Hallam University and is the author's own work apart from where indicated by specific reference to other sources. The work has not been submitted as part of any other award or presented to any other institution.

Alireza Janani 20/01/2018.

Acknowledgement

I would like to, firstly, thank my wife who supported me through thick and thin in this journey. I couldn't complete this program without her support and patience. My gratitude goes to my Director of Studies, Prof. Jacques Penders, for his continued invaluable support, advice and guidance and with whom my robotic life began. I am also grateful to Dr Lyuba Alboul, my Doctoral supervisor, who challenged my thinking and made me realise the importance of each small achievement. Both of you have been inspirational and this journey would not have been possible without you. Finally, I would like thank my sister and parents, who also supported me in this journey.

Contents

List of Figures

List of Tables

CHAPTER 1

AGRICULTURE AND ROBOTICS: TRENDS AND CHALLENGES

This chapter presents the research motivation and background, and the research aim and objectives. The main aim of this chapter is to assure that the reader comprehends the urge for human independent approaches to perform agricultural tasks. A global and epidemic issue in agriculture which jeopardizes the future of food industry is discussed in 1.1. The current trends aiming to compensate the raised issue are presented in 1.2. The direction of research and aim and objectives of the research are listed 1.3. Finally, the thesis layout is explained in 1.5

1.1 Agriculture: Everlasting Tension

Agriculture is defined as domestication of animals and plants. There are various tasks to consider in the field of agriculature. Chandrasekaran et al. (in 2010) classified agricultural branches into various categories including, Crop Production (dealing with production of various food crops), Horticulture (dealing with production of flowers, fruits, and vegetables), Forestry (dealing with large scale cultivation of perennial trees for supplying wood), Animal Husbandry (dealing with maintenance of various types of livestock for direct energy), Fishery Science, and Home Science. The archaeological excavations date back agriculture to 10,000 years ago. Since then, human beings have used agriculture mainly to provide food. However, nowadays agricultural products are demanded by various industries including pharmaceutical, energy and fashion. (Chandrasekaran et al., 2010).

As population increases, the demand for food increases too. Between 1960 and 2000 the population of the earth was doubled. In return, agricultural productivity improved by 2.4% annually between 1969 and 1989, but this fell to only 2% between 1989 and 2000 (Fao, 2002). By 2050, it is estimated that the population of the earth will reach 9.15 billion by which time the overall demand for agricultural products is expected to grow at 1.1% per year (Alexandratos and Bruinsma, 2012). To satisfy this level of demand, the agricultural productivity has to be improved by 25% of the current rate (Fao, 2009).

This everlasting tension has always forced farmers to invent more efficient cultivation methods. However, recent researches are suggesting that the future improvements may not be easily achievable. The main reason is the shortage of input labour force. In Japan, the number of farmers is decreasing, and every year more rice paddies are abandoned, while the average age of the farmers is increasing (Noguchi and Barawid, 2011). In addition, studies reveal that the younger generations are not interested in agricultural activities. According to Bloss (2014), 60% of the farmers in Japan are over 65 years of age.

The shortage in input labour force is a global issue, and it is not limited to a particular region. For instance, in United States of America, less than 1% of people are engaged in agricultural activities (Bloss, 2014), and in Europe (EU-27), 24.9% of agricultural labour dropped(Eurostat, 2015) since 2000.

The main reason is that agriculture is a labour intensive activity with mediocre to low level income. In addition, traditional and conventional farming methods, for example relying on seasonal rain for watering the crop, endangers the end products (Golait, 2007). In this era where demand for food is continuously increasing, the shortage of manpower makes achieving the targeted improvement in agricultural

methods much more difficult.

1.2 Agriculture: Current Trends and Methods

Over the years, more efficient tools and techniques have been developed to improve productivity of agricultural processes. However, the proposed methods require one common parameter: Human Labour. In recent decades, various approaches have been developed to minimise dependency to human input labour force. As a result, two main trends have been identified: (I) Mechanisation and (II) Precision Farming (Zhang et al., 2002).

1.2.1 Mechanisation

Mechanisation in agriculture refers to "the application of tools, implements, and powered machineries as inputs to achieve agricultural production" (Clarke, 1997). The main advantage of agricultural mechanisation is that the technology satisfies the real need of the farmers while the prices are affordable (Houmy et al., 2013). Generally, agricultural machineries can be powered from three sources of energy: manpower (or manual), animal, and motorised (using fossil fuel or electric power). A manpower source produces 0.1 Hp (Horsepower) over a limited period. However by harnessing power of animals, the productivity increases by 6 to 7 fold per animal. By invention of combustion engine, the productivity was increased even more, up to 10 fold. Today, agricultural tractors boost this efficiency up to 450 kW (612 Hp) or more (Shearer and Pitla, 2013). But, even with the introduction of tractors still a human is required for operation. A human has to drive the machineries through a large field for various purposes during cultivation period.

In open field farming, one popular trend to reduce field processing time is to process

larger portions of the field at a time. This requires the machines to become larger and heavier, and this will lead to occurrence of soil compaction.

Soil compaction occurs whenever an external stress or pressure exceeds the internal soil strength, also known as pre-compression stress value (Horna et al., 1995). Soils level of stress depends on the soil type, the climate, and varies from one location to another. Soil compaction results in reduction of soil pore volume which causes reduction in space for air and water in the soil, and consequently reduction in soil water infiltration (Graves et al., 2015). As a result, less rainwater can penetrate into the compacted soil which either increases the potential of erosion and runoff, or may cause the water to remain on the soil surface for longer period of time, especially in wheel tracks (Wolkowski and Lowery, 2008). Other impacts of soil compaction are limited root growth, reduction in nutrient uptake by roots, and reduction in micro-organism and earthworm activity (Graves et al., 2015). Recovering from soil compaction is time consuming, and it increases the cost of cultivation by 90% (Blackmore, 2012).

1.2.2 Precision Farming

Even though by mechanisation the productivity is boosted significantly, the need for a human as an operator constrains further progress. However in recent decades, progress in search for human independent approaches has led to invention of autonomous unmanned robotic vehicles.

The search for autonomous robots in agriculture started in 1920 when furrows were used to guide tractors across the fields with reduced effort from the operator (Mousazadeh, 2013), (Shearer et al., 2010). However the concept of fully autonomous agricultural vehicles dates back to the 1950s and 1960s where unmanned agricul-

tural vehicles navigated throughout the field using leader cable guidance (Ming et al., 2009). Today, autonomous agricultural robots are applied for various tasks. Grift et al. (2008) classified advancement in agricultural robotics into four main areas: (1) Plant Oriented Robotics, (2) Animal Robotics, (3) Controlled Environment Robotics, and (4) Field Robotics. Significant advances have been made in recent decades in each of these application areas.

Even though robotic systems are slow in speed, they can operate consistently with high precision for a long period of time in different weather conditions while reducing the labour cost. In fact, autonomous dedicated farming units increase efficiency and yield (Sukkarieh, 2012). This has convinced many countries to invest in autonomous agricultural approaches (Tarannum et al., 2015). For instance in 2014, the Japanese government announced plans to fund the development of unmanned farm tractors (Bloss, 2014).

Although unmanned autonomous robotic approaches have succeeded at reducing the labour force, they still require human observation for maintenance and failure recovery (Noguchi and Barawid, 2011). Moreover, the current autonomous tractors are costly, and only a narrow margin of farmers can afford them.

1.2.3 Multi Robot Approach

The idea of true human independent farming (not even as a supervisory role) led to the invention of a new approach: *multi robot system*. Multi robot system is a well-known topic in the robotics community in which a team of low-cost robots is deployed into the field to cooperatively execute the given task.

Multi robotic approaches have several potential advantages over single robotic approaches. With the use of multiple simple and low cost robots, the total execution

cost of the system can be reduced (Jones and Mataric, 2005). In addition, a team of robots can increase system flexibility and robustness by taking advantage of inherent parallelism and redundancy (Liu and Wu, 2001). Furthermore, multi robotic approaches have better system reliability and scalability (Yan et al., 2013). Finally, they have the potential to become completely human independent (Noguchi and Barawid, 2011).

However, the main drawback of this approach is system complexity. Deploying multiple robotic units, which do not interact with each other, in a field to execute the same task is insufficient, and often it results in complete failure (Jones and Mataric, 2005). The key parameter to an effective and successful team of robots is cooperation. Cooperation emerges from individual cooperative behaviour. Cooperative behaviour of robots has to be planned and designed accurately so that unwanted competition among robots is prevented (Jones and Mataric, 2005).

In order to have a cooperative team of robots, first characteristics of the team have to be identified. These characteristics define robots' limitations and capabilities, and sometimes are referred to as *group architecture* (Uny Cao et al., 1997). These characteristics are as follows:

- **Communication or Interaction Structure**: explicit vs implicit.
- **Control Structure**: centralised vs decentralised.
- **Differentiation**: homogeneous vs heterogeneous.

Next, three main questions have to be answered:

1. How to divide the complex global task into smaller manageable subtasks (i.e. task partitioning)?

2. How to distribute and assign each subtask to each participating robot (i.e. task allocation)?

3. How robots can attend their assigned task in a shared environment without any collision and congestion (i.e. coordination)?

Consequently, by answering these three questions, cooperation could emerge from a team of robots. The emerged cooperation can be scaled and quantified with the following parameters: *scalability*, *robustness*, *self-organisation*, and *coordination strength* (Barca and Sekercioglu, 2013), (Farinelli et al., 2004). These terminologies will be discussed in more details in Chapter 2.

The current trend in multi robotic approaches in the field of agriculture is to achieve cooperation through a dedicated coordinator via explicit forms of interaction. In explicit forms of interaction, each robot deliberately broadcasts its intentions to other members or an individual in the team. These messages could be transmitted via Wi-Fi, Bluetooth, or via any other types of signalling including blinking LED, ultrasounds, vibration, and etc (Uny Cao et al., 1997). With explicit forms of interaction, "robots' behaviours can be planned according to a complete prior knowledge" Barca and Sekercioglu (2013). However, these approaches are susceptible to the loss of communicating signal while the computational cost increases with the increase in the number of participating robots. In addition, they have limited scalability, robustness, and self-organisation (Parker, 1993) (Lumelsky and Harinarayan, 1997). However, in implicit forms of interaction, robots intentions are observed and interpreted either by changes in the environment created as a result of robots movement or task execution or local interaction among robots (Uny Cao et al., 1997). This form of interaction is unintentional, and hence it is referred to as indirect form of interaction (Barca and Sekercioglu, 2013). With implicit forms of interaction, the

system can become scalable, robust, and highly self-organised. Besides, the loss of communicating signal can no longer affect the success of the team.

1.3 Research Question, Aim and Objectives

To this date, there have been no approaches in the field of agriculture in which cooperation is achieved through implicit forms of interaction. This research is conducted to investigate **the feasibility of cooperation via implicit forms of interaction in a large team of robots for various agricultural tasks which are executed in an open field**.

The main aim of this research is **to develop cooperative behaviour mechanism by which a team of robots can execute an agricultural task (e.g. ploughing, spraying, and harvesting) in a large environment without the use of explicit forms of communication or central organiser.**

To obtain this aim, the following objectives have been foreseen:

- Description of the team architecture and the cooperative model.
- Review and analysis of related works.
- Analysis of the targeted tasks.
- Description and modelling of cooperative approaches.
- Simulation of the proposed approaches.
- Implementation and Validation of the designed approaches.

1.4 Thesis Contribution

Implicit Forms of Communication in Swarm in Open Field Farming

To this point, there have been no attempt in open field farming in which a cooperative behaviour emerges in a team of robots as a result of implicit interaction. All of the related works in the application of open field farming have been accomplished using explicit forms of communication. In this thesis, the proposed approaches are based on implicit forms of communication. In other words, robots do not transmit their intentions intentionally using explicit forms of interaction.

Spatial Congestion Clearance Using Implicit Form of Interaction In general, when operating in a shared environment, robots may aim to navigate to a shared coordinate in space. In this situation, congestion is inevitable. Normally, a central unit coordinates the robots to prevent congestion. Alternatively, a congestion is resolved between two robots using some form of signalling. Both approaches require explicit form of interaction. In this thesis, an approach is suggested that only relies on implicit form of interaction.

1.5 Thesis Layout

The layout of this thesis are as follows:

Chapter 1: This chapter presents the research motivation and background, and the research aim and objectives. The main aim of this chapter is to assure that the reader comprehends the urge for human independent approaches to perform agricultural tasks. In the first section, a global and epidemic issue in agriculture which jeopardizes the future of food industry is discussed. Next, current trends aiming to compensate the raised issue are presented. Finally, the direction of research and aim and objectives of the research are listed.

Chapter 2: In this chapter, the terminology introduced previously are discussed. Next, with the defined terminology, the architecture of the team which is considered in this thesis is introduced. Finally, similar works are described and analysed.

Chapter 3: In this chapter, ploughing with a team of robots is discussed. First, the task of ploughing is analysed to determine the design considerations and requirements. Next, the recognised problems are addressed, and two cooperation models are presented. Further, the results obtained from analysing these models are presented, compared, and criticised. Finally, to improve the system and compensate the identified drawbacks, an optimised method is presented, analysed, and compared with the previous cooperation models.

Chapter 4: In this chapter, spraying with a team of robots is discussed. This includes analysis and identification of design consideration and requirements of the task of spraying, comparison with ploughing, description of the cooper-

ative model, implementation and discussion regarding the obtained simulation results.

Chapter 5: This chapter presents conclusions and discussions over the obtained results. Moreover, it aimed to identify the future research axis to this research.

1.6 Research Location

The research is carried out in Sheffield Hallam University, Materials and Engineering Research Institute (MERI), Centre for Automation and Robotics Research (CARR). Numerous robotic researches have been conducted in this centre in variety of robotic fields from human robotic interaction to multi robotic systems and swarm robotics. The related projects in multi robotics and swarm robotics include GUARDIAN (Guardian, 2010) and I-SWARM (ISwarm, 2006). In the GUARDIAN project, a team of robots provide vital environmental information (including obstacles, particular toxic gas level, and etc) to a fire fighter who is surrounded by the team through the modified oxygen mask. On the other hand, the aim of I-SWARM is the realisation of collective intelligence of swarm of microrobots, in terms of cooperation and collective perception using knowledge and methods of pre-rational intelligence, machine learning, swarm theory and classical multi-agent systems (ISwarm, 2006). In addition to swarm related projects, the research centre hosts projects from other areas in the field of robotics. In human robot interaction area, CARR hosts Engineering and Physical Sciences Research Council (EPSRC) funded project called REINS (REINS, 2011). In REINS, the focus is to develop a semi-autonomous mobile robot with sensory capabilities that transfer environmental information to the attached fire fighter via haptic and tactile interaction methods in a smoked filled environment in which the visibility is as low as none.

The supervisory team, Prof Jacques Penders and Dr. Lyuba Alboul, were involved in the mentioned projects along with other related research projects which are not mentioned in here.

CHAPTER 2

LITERATURE REVIEW

The main aim of this chapter is to demonstrate that the current approaches in the agricultural robotics are not reliable and sufficient for future development and expansion. The relevant definitions in multi robot system are discussed in 2.1 and 2.2. The application domain in multi robot system is reviewed in 2.3. Multi-robotic team characteristics are reviewed in 2.4 and design questions in any multi-robotic team is mentioned in 2.5. Finally, the previous contributions in multi-robotic farming is reviewed in 2.6.

2.1 From Multi Agent System to Multi Robot System

The modern approach to artificial intelligence is centred around the concept of autonomous agents (Vlassis, 2003). According to Wooldridge and Jennings (1995), an entity can be counted as an agent if it has the following properties:***autonomy***: to make decisions without the direct intervention of others, ***reactivity***: to perceive the surrounding environment (e.g. the physical world or a graphical user interface) and to provide appropriate response. ***pro-activeness***: to exhibit goal-directed behaviours by taking the initiative. ***social abilities***: to interact with other entities via some kind of agent-communication language. Inter-agent communication has utmost importance in multi agent systems in which multiple autonomous agents aim to solve a problem cooperatively.

The concept of an agent in computer science, generally is applied to software as it entails all the aforementioned characteristics. However, robots can be counted as agents too since robots perceive their surrounding environment through sensors and

act upon that environment through actuators in a goal-directed behaviour while they can communicate with other robots in the environment (Russell and Norvig, 2003). Therefore, the existing concepts and solutions in agent-based systems can be applied (if applicable) to robotic-based and multi robotic-based problems.

2.2 Multi Robot System

In the past decades, single robot systems have been applied to numerous application domains. As tasks become more and more complex, the robotic units are designed more complex to fit the given task. However, there are tasks which a single robot, regardless of how sophisticated the robot design is, is either incapable to carry out successfully or it requires a long time to complete the process. Consider a search and rescue scenario in which a robot is given a map of the environment and it is required to search for injured human beings and to act appropriately upon detection. Clearly, time is an important asset in this example, and it is preferred to perform the task as fast as possible. If a single robotic unit is deployed, despite how fast it performs at every single point on the map, the maximum efficiency cannot be reached. In addition, the success of the task is susceptible to the loss of the single robot. In other words, if the robot fails in the middle of the operation, the task cannot be completed and the robot has to be recovered.

These weaknesses of single robotic units have led to the deployment of multiple robots. In this approach, a group of robots functions together to complete a shared goal. The key to success of the given task is the presence of some form of cooperation among the individuals in the group. In other words, it is not possible to achieve success within a group of non-cooperative single robotic units.

2.3 Application Domain of Multi Robot System

The concept of multi robot systems is being applied to various new tasks every year. In this section, we provide a brief overview of the current application domains for multi robot systems.

2.3.1 Foraging

In foraging, the aim is to pick up and gather objects which are scattered in the environment (Farinelli et al., 2004). This application domain is inspired by the behaviour of ants that search for food sources distributed around their nest Sahin et al. (2008). The main challenge in this domain is to implement an optimised search strategy to maximise the ratio of the returned food. There are various tasks which require foraging including search and rescue (Beck et al., 2016), toxic waste cleaning, mine cleaning, and service robots (e.g. (Jung and Zelinsky, 2000);(Jeon et al., 2016)).

2.3.2 Area Coverage and Exploration

Similarly to foraging, in area coverage, the aim is to visit or analyse all the free points in space as efficiently as possible (Choset, 2001). Various tasks including demining (Santana et al., 2005), snow ploughing (Saska et al., 2013), line searching (Marjovi et al., 2010), open field processing (Noguchi et al., 2004); (Batalin and Sukhatme, 2002), beacon distributing (Howard et al., 2002), lawn mowing (Zheng et al., 2005), and car-body painting (Graca et al., 2016) are categorised in area coverage application domain.

In a slightly different area of application, robots are aimed to explore and analyse an unknown environment. This task is also referred to as exploration. The main

difference between exploration and area coverage application domains is that in area coverage the map of the environment may or may not be given to the robots prior to execution, whereas in area exploration the field is completely unknown to the robots. Therefore, the main issue in area exploration is to generate a global map of the environment cooperatively and use the map for further navigation or processing (e.g. (Simmons et al., 2000); (Thrun and Liu, 2005); (Koch et al., 2015)).

2.3.3 Multi-Target Observation

In multi-target observation, also known as CMOMMT: Cooperative Multi-robot Observation of Multiple Moving Targets (Farinelli et al., 2004), first introduced by Parker (1999), a team of robots are required to detect and track a group of moving targets cooperatively. Multi-Robot Target Observation has many connections with security, surveillance and recognition problems (Werger and Mataric, 2000) where targets moving around in a bounded area must be observed.

2.3.4 Object Transportation

In this application domain, robots are required to transport objects from point A to point B. Depending on the size of the objects and capabilities of the robots, the approaches could further be classified into (i) cooperative pulling/pushing and (ii) individual object transportation. In cooperative pulling/pushing, individuals are incapable to transport an object from A to B. Instead, they cooperatively push or pull (sometimes the combination of both) the object (e.g. (Mataric et al., 1995);(Jose and Pratihar, 2016);(Weber et al., 2015);(Vig and Adams, 2006);(Yamada and Saito, 1999)). In object transportation, however, the robots are capable to pick up and carry an object from one location and deliver it to another location individually (e.g. (Wu et al., 2016a); (Barrientos et al., 2016)).

2.3.5 Flocking

In the flocking task, the goal is to navigate together while particular formation is maintained in the team. This application domain is inspired by formation control of a flock of flying birds. Cooperation among the individuals is also used to localise each other, and to fuse information acquired from the environment. Similar to exploration, map building of unknown environments is a common task tackled in this application domain, though the difference is that in exploration, there may or may not be a particular formation among the team. The problem of exploration and flocking is related to several applications such as transshipment operations in harbours, airports and marshalling yards (Arbanas et al., 2016), motion coordination in industrial applications and exploration of dangerous environments (Solovey et al., 2015).

2.3.6 Soccer

In recent decades, robotic soccer has become an interesting test bed for research in cooperative multi agent and multi robot system (Kitano et al., 1997). This is because the environment in which the robots operate is dynamic, and hostile by which coordination becomes extremely challenging (Farinelli et al., 2004).

2.4 Team Characteristics

Before proposing any design or approach, the characteristics of the team have to be identified. These characteristics determine the individual and the team level control structure, strategies or steps to distribute the subtasks and to allocate the required resources. In this section, important team characteristics are reviewed.

2.4.1 Control Structure

The most fundamental decision that has to be made in defining characteristics of a multi robotic team is how individuals in the team perform decision making. Decision making is referred to as a "cognitive process resulting in the selection of a course of action among several alternative scenarios" (Yan et al., 2013). In a multi robot system, decision making is carried out in *centralised* or *decentralised* manner.

Centralised: In a centralised team of robots, there must be at least one robot or computer which has a complete global information of the environment, other robots, and the state of the task. This centralised unit is needed to perform task allocation and coordination among the participating robots in the team (Barca and Sekercioglu, 2013).

The centralised approach has been studied extensively in various fields of application. Tang and Parker (2005) developed a centralised based system by which a collection of heterogeneous robots reorganise into subteams as needed depending upon the requirements of the application tasks and the sensory, perceptual, and effector resources available to the robots. Tang and Parker then applied the system to the task of box pushing and object transportation. The reason that robots have to form particular formation at the beginning of the task is that not all robots in the team have localisation system. The formation will assist the robots without localisation to navigate. Khan et al. (2016) proposed a centralised system for formation control of nonholonomic mobile robots for obstacle avoidance in a cluttered environment. Khan et al. tackled the problem by using "proportional-integral average consensus estimators", whereby information from each robot diffuses through the communication network. Banfi et al. (2016) provides another example of centralised

control for multi robot exploration application. In this approach, robots connect to the base station only when making new observations so the communication is performed more effectively. In exploration in known environment application, Yan et al. (2010) developed an online sampling-based graph, by which the optimal path is calculated for the robots. Wurm et al. (2008) proposes a coordination mechanism for a team of exploring robots using segmentation of the environment to determine exploration targets for the individual robots. The central unit assigns each robot to a separate segment, thus a balanced distribution of the robots over the environment is achieved. Yan et al. (2012) developed an empirical-based heuristic planning strategy for the goods transportation by multiple robots. In here, the focus is to plan the transportation task for each robot by estimating the production rate of goods based on multi-robot coordination. The centralised approach is also used for multi robot path planning. Luna and Bekris (2011) present an efficient and complete approach for multi-robot path planning problems using graph-theory. In there, the central unit performs the calculations and assigns the paths to each robot.

Decentralized A decentralized architecture is categorized as either distributed or hierarchical (Uny Cao et al., 1997). In a distributed approach, there is no central agent to perform decision making, and individuals are equal from the control the point of view, and each participating individual performs decision making completely autonomously. In a hierarchical architecture, robots are divided into smaller groups, and decision making in each group is carried out by local decision makers (Uny Cao et al., 1997). Yan et al. (2013) refers to this approach as hybrid since it has properties of both the centralised and decentralised approaches.

Decentralised systems are applied in various multi robot applications. In search and rescue, Penders et al. (2011) developed a cooperation mechanism for a team

of homogeneous robots by which a fire-fighter is guided through a smoked field environment. In this example, robots always maintain their distance toward the fire-fighter and each other. All the decision making is carried out individually and robots utilise only the information collected by their sensors. In area coverage application domain, Ranjbar-Sahraei et al. (2012) presents an intelligent approach by which a team of robots covers an area cooperatively. In this example, robots mark their territory by dispensing pheromone-like materials at the border of the claimed area. Upon detection of another robots' pheromone, they perform simple navigational movement (e.g. turn left/right). In formation control application domain, Lopez-Gonzalez et al. (2016) proposes a formation scheme, based on Lyapunov techniques, if the orientation and distance information for each robot is available locally.

Discussion Although, with a centralised system, an optimal plan based on the global knowledge can be planned, the efficiency drops significantly as the size of the team increases. Besides, it is not robust in relation to dynamic environments or failure in communications and other uncertainties (Yan et al., 2013). Moreover, a centralised architecture is a leader dependent approach which means with failure of the central unit the system will be incapable of operating (Parker, 1993).

On the other hand, using a decentralized system carries advantages including the decrease in delay and impracticalities associated with centralized processing, independence of computational complexity and size of the team, increase of robustness toward the loss of the leader, and efficient use of parallelism. The only drawback of a decentralized architecture is limited awareness of individuals about the global knowledge (Barca and Sekercioglu, 2013).

2.4.2 Differentiation

It is important to decide whether the individuals in a team of robots have the same capabilities or not since this will greatly affect how the underlying control schemes will operate (Barca and Sekercioglu, 2013).

A team of robots could be either homogeneous or heterogeneous. In a homogeneous team, individuals are identical meaning that they have the same hardware and control software (Uny Cao et al., 1997). Various examples involving homogeneous team of robots are in cooperative path planning (Habibi et al., 2016), cooperative localisation (Tsai et al., 2015), search and rescue (Balta et al., 2015); (Couceiro, 2015), cooperative exploration in an unknown environment (Wang and Olson, 2016), and target tracking (Zheng and Tan, 2015); (Senanayake et al., 2016).

In a heterogeneous team of robots, individuals have different designs and functionalities which compliment each other Barca and Sekercioglu (2013). Like homogeneous teams of robots, heterogeneous teams of robots have been deployed in various areas of application including topological map-building (Ramaithitima et al., 2016), search and rescue (Beck et al., 2016);(Gunn and Anderson, 2015), cooperative area coverage (Pierson and Schwager, 2016), formation control (Sen et al., 2016), cooperative exploration in an unknown environment (Dai et al., 2016), foraging (Castello et al., 2016); (Prorok et al., 2016),and target tracking (Robin and Lacroix, 2016).

It is important to note that heterogeneity introduces more complexity to the system due to (1) highly complex task allocation and (2) more dependency on modelling other robots in the team (Uny Cao et al., 1997). Individuals in a heterogeneous team of robots have different capabilities. To quantify this, a concept called *task coverage* is introduced. Task coverage (in individual level) measures "the ability of an agent or a member of a team to achieve a given task" (Parker, 1994a). At the team level, task

coverage determines the level of cooperation in the team. When the task coverage is high, task can be accomplished without cooperation, but otherwise, cooperation is necessary. In the homogeneous teams of robots, the task coverage is maximal and decreases as the group becomes more heterogeneous. Unlike homogeneous, in the heterogeneous team of robots task coverage is determined based on individual capabilities hence more parameters should be considered in the calculation of task coverage (Uny Cao et al., 1997).

One common trend for task allocation in a homogeneous team of robots is through role assignment which will be allocated either at design-time or arises dynamically in real-time (Uny Cao et al., 1997). Numerous researches have been conducted in this field and different task allocation and role assignment algorithms and techniques have been developed for homogeneous teams of robots including Contract Net Protocol (Smith, 1980), Dynamic token generation (Cottefoglie et al., 2004), Game theoretic approach (Arslan et al., 2007), etc. There are also approaches which do not rely based on any well-known algorithm. Mendoza et al. (2016) applied role assignment-based task allocation to a homogeneous team of robots deployed in ROBOCUP 2015 soccer competition. In this particular example, robots task, which is covering particular zone in the field, is carried out by a dynamic zone selection algorithm in which robots select specific zones in the field according to the flow of the game.

But is it possible for a team of homogeneous robots to turn into a heterogeneous one? In theory, a homogeneous team of robots is a team in which individuals are identical in terms of software and hardware. However, in various scenarios in which robots have learning and self-designing algorithms (e.g. (Li et al., 2002), Luke et al. (1998)) or structure reconfiguration capabilities (e.g. (Murphy, 2000)), with the

suggested definition, a homogeneous team becomes heterogeneous. In this scenario, the task allocation could be carried out in both ways depending on the application.

2.4.3 Communication Structure

Another important characteristic in any team of robots is how the team share their knowledge about the task and the environment. This is referred to as communication structure or interaction structure. Over the past decades, various classifications are proposed. Uny Cao et al. (1997) classified multi robot communication into three categories: (i)*interaction via environment*, (ii) *interaction via sensing*, and (iii) interaction via communication.

Later, a more abstract classification was used to describe interaction models. In this classification, interactions are either *implicit*, which encompasses interaction through sensing other robots and through the shared environment, or *explicit*, which encompasses interaction through communication, (Yan et al., 2013).

Interaction via environment In this approach, robots utilise the available hints in the shared environment created by other robots as a result of their execution to extract necessary information. This form of interaction is inspired by the complex nest building behaviours in ants and termites first described by Grassé (1959). Moreover, since the information is perceived and not intentionally transmitted, this form of interaction is categorised as an *indirect* form of interaction.

This form of interaction is extremely popular in the multi robotic and swarm community, and there are several examples in which cooperation among robots is merged as a result of their indirect interaction. Parker et al. (2003) presents a robust algorithm for collective robotic construction. In here, the robots, which are equipped with a force sensitive plough and collision sensors, aim to clear a designated field

of any rubbles with an algorithm called blind bulldozing. In this three state finite state machine algorithm, the robots wander around the field and plough the existing rubbles into the field borders. When the material that the robot is pushing exerts a force which is beyond certain threshold, the robot reorient its heading and continues as before. The similar reorientation occurs if the robot collides with another robot. Willmann et al. (2012) presents another scalable and robust cooperative team using only indirect form of communication. In this example, the autonomous flying vehicles are given the blueprint of the structure and the location where they can pick the building materials. In this sequential construction, robots place the building materials right next to the last placed building block. In this way, robots interaction is through the constructed building.

In (Ranjbar-Sahraei et al., 2012), a team of robots mark their territory by dispensing particular pheromone. Upon detection marks of another robot, the robot deviates its path. In this way, robots can cover the entire area cooperatively. Zedadra et al. (2016) demonstrates a multi agent foraging algorithm named Cooperative Switching Algorithm for Foraging (C-SAF) inspired from the classical ant system. In this approach, robots, while searching for food, mark their trail by dispensing detectable forms of pheromones. In the meantime, "robots create simultaneously and synchronously a pheromone wave front expansion from the nest to the food and can use the negative gradient to go back to nest"(Zedadra et al., 2016).

Interaction via sensing In an alternative approach, the necessary information is perceived by sensing other robots in the team. This approach is inspired by formation control in flocking birds described by Reynolds (1987). This form of interaction is also classified as an indirect form of interaction since there is no intention in conveying the information. One key requirement in this approach is that each robot has

to differentiate and recognise other robots from other objects in the environment which demands a modelling. The main application is those that require some form of formation control for example cooperative guidance of a firefighter in smoke filled environment (Saez-Pons et al., 2010). Saez-Pons et al. presents a cooperative team of robots that surrounds the firefighter and provide environmental information back to the firefighter. In this approach, robots aim to maintain their formation while navigating using the attached range sensors. In (Saska et al., 2013), the robots, which are performing snow ploughing the roads at the airport, monitor each others' behaviour and trail with the attached range sensor to adjust their trajectory.

Interaction via communication In this approach, robots transmit their intentions directly to another robot or broadcast via explicit forms of communication including RF messages (e.g. Bluetooth, Wi-Fi), visual signals (e.g. blinking LED, RGB LEDs), and other methods including ultrasound messages (Uny Cao et al., 1997). This often requires a dedicated onboard communication module (Yan et al., 2013). An interesting taxonomy based on communication presented by Dudek et al. (1996) classifies multi robotic teams according to their communication range, communication topology, and communication bandwidth. Since conveying information is intentional, this form of interaction is referred to as a direct form of interaction. Extensive works have been carried out in this form of interaction in various application domains. Simmons et al. (2000) demonstrates how a team of robots can cooperatively create a global map of the environment. In this example, the mapping problem is decomposed in a modular, hierarchical fashion: Each robot maintains its own local map. A central module receives the local maps and combines them into a single, global map. Rao et al. (2016) presents an algorithm for cooperative exploration in which robots exchange the required information with each other rather

than a central module. In this approach, Utility Based Return (UBR) is used to perform collaborative exploration. All vehicles start moving from a unique base station and spread out by performing frontier-based exploration. Frontiers are selected based on a utility value which is a trade-off between the expected information gain at the frontier and the cost to get to it. The vehicles in the vicinity of each other assign these frontiers among themselves based on the utility value so as to maximise the total utility of the exploration. The vehicles sense obstacles and map the area according to the obstacles found, while detecting new frontiers to decide where to move. When in range with other vehicles, the exploration decision is collaborative and each vehicle chooses a different frontier to explore in order to avoid more than one vehicle exploring the same area.

Discussion Although with explicit forms of communication high level of accuracy can be achieved, the overall computational cost of the system increases as the number of participating robots increases. With implicit forms of communication, although perception and interpretation of information are more complex, the stability, flexibility, and fault tolerance of the system are better than with explicit forms of communication. However, various researches suggest that systems in which interaction is carried out by the combination of both implicit and explicit forms of communication simultaneously can have both approaches' advantages. Various pieces of literature refer to this form of interaction as a hybrid form of interaction including (Barca and Sekercioglu, 2013);(Yan et al., 2013);(Sahin et al., 2008) to name a few.

2.4.4 Representative Architecture

Any multi robot system inevitably needs to define the aforementioned parameters prior to system level design or individual behaviour design. In fact, these axes determine the type of behaviours required for individuals. For instance, if a team of robots is homogeneous, individuals can model each with unified dimensions. Over the past decade, several architectures have been developed for different applications. Few of these architectures are so well-defined that they are adopted as sub-categories and directions in multi robot system. In this section, few of these architectures are reviewed.

CEBOT (Cellular roBOTics system): First introduced by Fukuda and Kawauchi (1990). It is a decentralised hierarchical architecture. A CEBOT team is divided into smaller groups with local leaders. The leader robots are responsible to allocate subtasks and communicate with other leaders. A distinctive characteristic of CEBOT is that robots dynamically reconfigure their formation by changes in the environment as the architecture is inspired by the cellular organisation of biological entities (Cao et al, 1997).

SWARM: It is a distributed, decentralised, autonomous large groups of robots ($n > 10$) that normally interact with each other through the environment. Swarm robotics inspired mostly by swarm intelligence. Swarm intelligence is defined as collective intelligence that emerges from interactions among large groups of autonomous individuals (Barca and Sekercioglu, 2013). Sahin (2004) defines "Swarm robotics as the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among the agents and between the agents and the environment". In swarm robotics, the main inspirations stem from the observation of social insects (Sahin

et al., 2008). Robustness, flexibility, and scalability are built in characteristics of a swarm.

GOFER: It is first developed by Caloud et al. (1990) to study distributed problem solving in indoor environment in a team of mobile robots (Uny Cao et al., 1997). In this approach, a central unit, which has a global view of the given task and the rest of the robots, generate a plan structure which contains task distribution and team scheduling. The GOFER architecture was successfully used with three physical robots for tasks such as following, box-pushing, and wall tracking in a corridor.

ACTRESS (ACTor-based Robot and Equipment Synthetic System): First introduced by Asama et al. (1989), and inspired by Universal Modular ACTOR Formalism (Hewitt et al., 1973). This architecture consists of three heterogeneous robots each responsible for different task along with three workstations. In the deployed application domain, individuals are not capable to perform the given task (e.g. pushing a box) alone, and they have to cooperate with each other. The cooperation and coordination are achieved through negotiation among robots, hence the focus of this approach is to improve the efficiency of the communication among robots.

ALLIANCE: Introduced by Parker (1994b), ALLIANCE is developed originally to study cooperation in a heterogeneous, small-to-medium-sized team of robots. ALLIANCE is a hybrid in communication architecture since the information is perceived from the environment as well as broadcast messages in the environment.Although with explicit forms of communication high level of accuracy can be achieved, the overall computational cost of the system increases as the number of participating robots increases. With implicit forms of communication, although perception and interpretation of information are more complex, the stability, flexibility, and fault tolerance of the system are better than with explicit forms of communication. How-

ever, various researches suggest that systems in which interaction is carried out by combination of both implicit and explicit forms of communication simultaneously can have both approaches' advantages. Numerous pieces of literature refer to this form of interaction as hybrid (Barca and Sekercioglu, 2013);(Yan et al., 2013);(Sahin et al., 2008).

2.5 Main Questions in Multi Robot System

The multi robot system is a group of autonomous agents which have a shared goal and work together to achieve the given task. From the psychological point of view, robots are selfish, utility-driven agents (Uny Cao et al., 1997) which only aim to succeed at the given task. In this situation, competition rather cooperation emerges in a group of single robot systems to access the shared resources in the environment (Jones and Mataric, 2005). It could be said that the key element that separates multi robot teams from a group of single robot systems is cooperation (Jones and Mataric, 2005).

Cooperation requires three main elements: *task partitioning*, *task allocation*, and *coordination*. In any multi robot application, the global task has to be analysed, and if necessary the global task has to be divided into manageable portions referred to as sub-tasks. This process is referred to as *task partitioning*. Next, each subtask is distributed to one or more individuals in the team. This process is referred to as *task allocation*. Finally, robots attending sub-tasks require resources (including a point in space) which have to be accessed while any probable congestion is avoided. This process is referred to as *coordination*. In this section, these research elements and current trends to resolve them are reviewed.

2.5.1 Task Partitioning and Allocation

The problem of task allocation is to respond to the question of *"which robot is doing which task?"*. In order to answer this question, the global task has to be decomposed into two or more sub-tasks. Task allocation and task partitioning are strongly intertwined together. According to Dias et al. (2006), there are two common approaches for the aforementioned question. In one approach, the task is first decomposed and then sub-tasks are allocated to the robots (also refer to as decompose-then-allocate) (Caloud et al., 1990), or the global task is given to all individuals, and each robot individually decomposes it to smaller manageable pieces (also known as allocate-then-decompose) (Botelho and Alami, 1999);(Pini et al., 2011).

Task partitioning (or decomposition) also depends on the characteristics of the task and the capabilities of the robots. According to the taxonomy provided in (Gerkey and Mataric, 2004), a team of robots and a task can be classified based on few characteristics:

(I) Single Task robots versus Multi Task robots: Multi Task refers to robots which are capable of executing multiple tasks at a time, and Single Task refers to robots which are capable of executing only one task at a time.

(II) Single Robot task versus Multi Robot task: A Single Robot task refers to a task which require only one robot, whereas a Multi Robot task refers to a task which require more than one robot to be completed.

(III) Instantaneous Assignment versus Time extended Assignment: Instantaneous Assignment means that the task can be allocated to individuals instantaneously and without any extra information. Time Extended refers to tasks in which task allocation demands more information regarding the envi-

ronment, the team, or the completion state of the task.

There are various methods that the global task can be divided and distributed among a team of robots. In here, the reviewed task allocation methods are divided into two main approaches: (I) Static Task Allocation, and (II) Dynamic Task Allocation.

In static task allocation, prior to task execution, the task is divided into smaller manageable sub-tasks and distributed to each individual in the team at the design time. For instance in applications related to processing an open field, the environment is first divided and distributed among robots in a way so that robots' navigation are minimised using various theories and algorithms including graph theory (Sungjun et al., 2015), (Fazeli et al., 2010), and (Ahmadi and Stone, 2006). Although static allocation is fast and efficient, it does not tolerate any real-time changes that could occur in the environment.

In dynamic task allocation, the task will be distributed among robots during the task execution. According to Karla and Martinoli (2006) The dynamic task allocation can further be classified into two sub-categories: threshold-based and market-based task allocation. In threshold-based approaches, the robots utilise the changes in the environment to identify the state of the global task and what is required to be done next. A robot resumes its search for hints in the environment until it reaches the conclusion that the global task is completed. Threshold-based task allocation has been applied to various application domain including foraging (Krieger et al., 2000) and aggregation (Agassounon and Martinoli, 2002).

In market-based approaches, robots act as self-interested agents in pursuit of individual profit. Tasks often are distributed through auctions held by an auctioneer. The auctioneer is either a supervisor agent or one of the robots. "Each robot locally plans the achievement of the available task, computes its cost of execution,

and encapsulate the cost in the bids" (Dias et al., 2006). Depending on the task, the auctioneer selects the highest or the lowest bidder (Karla and Martinoli, 2006). Sometimes, instead of allocating particular task, such as shoot the ball or cover a specific area, roles are assigned to the robots. Roles define a collection of related actions or behaviour (Dias et al., 2006). Auction-based task-allocations have been applied to cooperative exploration (Zlot et al., 2002) and object manipulation (Gerkey and Mataric, 2002).

2.5.2 Coordination

Coordination is the core element of a multi robot system by which the overall performance of the system is affected directly (Yan et al., 2013). Various works have been carried out in this area. In here, we review the available two important categories of coordination systems in multi robot systems: Dynamic and Static.

Dynamic vs Static In one classification, coordination can be carried out statically, or dynamically. In static coordination, which is also known as deliberative or offline coordination (Todt et al., 2000);(Iocchi et al., 2000), robots adapt series of conventions prior to engaging in the task (Yan et al., 2013). Kato et al. in (Kato et al., 1992) coordinates a team of robots by simply applying few navigational rules (e.g. "keep right" or "stop at an intersection", etc). Although coordination can be achieved efficiently and quickly, it requires detailed analysis of the given task, and it can easily fail by radical changes in the given task or the environment.

Dynamic coordination, which is referred to as online or reactive coordination (Todt et al., 2000);(Iocchi et al., 2000), is carried out during execution of a task and is based on "analysis and synthesis" of information which can be obtained by means of communication (Yan et al., 2013). As discussed in section 2.4.3, information among

robots is conveyed explicitly or implicitly. Accordingly, dynamic coordination is performed either by transferring intentional messages (i.e. explicit coordination (Gerkey and Mataric, 2004)) or as a result of robots local interacting with the other robots or the shared environment (i.e. implicit coordination). Dynamic coordination can well adapt to changes in the environment in real-time, however, coordination is becoming more complex by the increase in complexity of the task.

Taxonomy Based on Individual Knowledge and Awareness In another classification introduced by Farinelli et al. (2004), coordination in the team can be categorised according to individuals' knowledge and awareness. In this classification, a multi robot system is one of the following:

Unaware in which individuals have no information about other robots in the team while they execute their share of the task. Since robots have no knowledge about each other, the communication among the robots cannot be direct. Unaware teams are easily scalable, thus they are adopted for swarm robotic applications.

Aware, not coordinate systems in which robots of the team have the knowledge of the presence of other robots in the environment, and act together in order to accomplish the same global goal. However, a robot may not take into account the actions performed by other robots in order to accomplish its task.

Weakly coordinated systems in which individuals follow explicit predefined coordination protocols. In here, coordination protocols refer to the explicit predefined rules that select particular

Strongly coordinated, strongly centralized systems in which a robot plays the role of the leader and it coordinates other robots in the environment. This method requires explicit messages to be transferred among robots. This method suffers from robustness since the coordination is susceptible to the loss of communicating signal.

Strongly coordinated, weakly centralized systems in which the leader is selected in prior, and it will be selected during execution time. In weakly centralised approaches, the interaction architecture is hybrid. This means that the robots exchange information via explicit and implicit forms of communication. Noreils (1993) and Simmons et al. (2001) are two examples in which the coordination is achieved by means of hybrid interaction architecture.

Strongly coordinated, distributed systems in which each individual in the team executes a coordination protocol, and takes its decision completely autonomous. These systems are generally more robust to communication failures and robot malfunctioning, even though these problems may affect the overall performance of the team in the accomplishment of the task. "The strongly coordinated distributed approach entails that some kind of communication has to be used, and leaves unconstrained the other System Dimensions" (Farinelli et al., 2004).

2.5.3 Congestion Avoidance and Clearance

The goal of coordination is to prevent or resolve probable congestions during task execution. There are two sources of congestion in a multi robot system: collisions and resource conflicts. Resource conflicts occurs when multiple requests targeting the same resource arrive simultaneously (Yan et al., 2013). The problem of resource conflict is not specific to robotic applications. In distributed computing and multi-access networking, where a single resource is accessed by different processes, a resource conflict occurs (Uny Cao et al., 1997).

In multi robot system, a resource conflict arises when two or more robots need to access a shared space, communication media, or to manipulate an object (e.g. object transportation). The current trend to avoid or resolve resource conflicts is

to coordinate robots (Yan et al., 2013). As mentioned in section 2.5.2, robots in a team can be coordinated statically, or dynamically. In static coordination, probable congestions are avoided from the beginning of the task by "adoption of a convention" (Yan et al., 2013). Kato et al. in 1992 coordinate a team of robots to avoid collision and congestion by simply applying few navigational rules (e.g. "keep right" or "stop at the intersection", etc).

In dynamic coordination, robots resolve their congestion avoidance during execution time. Various approaches have been applied to resolve resource conflicts in a team of robots during execution time. However, the proposed approaches either aim to resolve the resource conflict via a central decision maker or by letting robots negotiate. In central approaches, the central unit, which has the global view of the environment, observes the trajectory of the robots and plans the trajectories of the robots so that any deadlock is prevented. This approach is inspired by the flight controls at the airport. At the airport, the flight control, which has the global position of each airplane, controls the landing and takeoff of each airplane to prevent collisions.

In negotiation-based methods, robots, at the point of conflict, exchange information to resolve the congestions. The decision making of the robots and further motion planning are completely decentralised and in parallel. Jager and Nebel in 2001 described a decentralised method to resolve a spatial resource conflict by which robots exchange information about their planned trajectories whenever the distance between two robots drops below a certain value. Using the exchanged information, robots can determine whether they are in danger of a collision/congestion or not. In case of detection of a possible collision, robots monitor each others trajectory, and if necessary they introduce time delays in certain points of their movements.

On the other hand, if a deadlock is detected, each robot plans different trajectories until the congestion is resolved. Marcolino and Chaimowics in 2009b proposed another method for avoiding congestion and collision in a swarm of robots moving in opposite direction. In their method, each robot perceives the possibility of collision and warns their teammates through local sensing and intentional communication. Upon receiving the warning, the following robots change their trajectories to avoid congestion. In another research (Marcolino and Chaimowicz, 2009a), a coordination algorithm is proposed for a scenario in which robots try to access the same target. The algorithm defines an action based on a probabilistic finite state machine and relies on the local sensing and communication.

The current trend in multi robot system and swarm robotic is to resolve the congestion via explicit forms of communication. To the best of our knowledge, there is not any systematic approach which relies only on implicit forms of communication to resolve resource conflicts.

2.6 Cooperative Farming: Review and Analysis

Although multi robot system and swarm robotic have been around for decades, these topics are still fresh in agricultural robotics. Agricultural tasks, particularly those related to open farms, have similarities with the previous area of application covered in swarm robotics and multi robot systems. Hence, many of the experienced gained in these examples can be used for cooperative farming. Liekna and Grundspenkins (2014) match the areas of application in swarm robotic and multi robot system with the task of cereal cultivation. In here, the author suggests that a team of robots, which are capable of harvesting and transportation, are required to:

aggregate : to achieve the starting point of the mission.

form patterns, self-deployment, area coverage : to effectively cover the entire field.

self-assemble : with the transporter.

object clustering and assembling : to mow cereal to a particular point of the fields, and to store goods effectively in the warehouses.

forage : to go into the field, and bring back the goods from the field.

In this section, a short summary of the tasks considered in this thesis is provided, and similar application domains and questions are identified. Next, the environment in which the robots are assumed to be operating is described. Finally, the related works are reviewed.

2.6.1 Agricultural Tasks

In multi robot systems, it is necessary to analyse the global task before any further development is performed. Task analysis helps to identify design questions, requirements, and solution boundaries.

Tasks in open field agricultural robotics can be categorised in two classifications; (1) Those in which the field can be processed from different points at the same time, hereafter referred to as *independent tasks*. (2) Those in which the field has to be processed sequentially, hereafter referred to as *sequential tasks*. In independent tasks, the result of execution of one robot will not affect others.

In this thesis, three agricultural tasks are considered: ploughing (or seed-bed preparation), spraying (of chemicals), and harvesting. These tasks are different in nature and require different cooperative mechanisms. In the task of ploughing, the field is processed sequentially, and in a particular order. This is because the result of

execution of the current part of the field depends on the results of the previous ones. In the task of spraying, the field is processed independently, meaning that the field can be processed from both directions and from different locations in the field. However, it has to be carried out in a restricted number of times. In harvesting, the redundancy in execution does not affect the results, and the field can be processed in any order. Other tasks including seedings, seed mapping, and etc belong to one of these three categories, and they can utilise the implemented approaches. Even though all tasks are executed within the same framework, each task has different properties and characteristics, hence different cooperative mechanisms are required.

2.6.2 Analysis of the Related Works

As mentioned, there are few examples in which agricultural tasks are tackled by a team of cooperative robots. One example of such system is RHEA (Robot Fleets for Highly Effective Agriculture and Forestry Management) European project (RHEA, 2014). The RHEA project is aimed to improve product quality by diminishing chemical usage in weed control (Drenjanac and Tomic, 2013). During execution, robots have to continuously report their status and position to the base station. At the base station, the transmitted status and position of the robots are analysed. Once all the required information is received, area decomposition algorithm divides a working area into cells which are then dynamically assigned to robots (Drenjanac et al., 2014). The RHEA presents a robust and scalable system, however, one main issue of this approach is that robots have to be within the communicating signal and that will limit the area that the team can cover during execution.

Anil et al. (2015) presents another swarm system for agricultural application. In this example, low cost multi-functional team of robots are developed. Each robot is

capable to reconfigure itself so that it can carry out different agricultural processes such as ploughing, seeding, spraying, and harvesting. Robots are equipped with appropriate tools to carry out each task. In addition, each robot is also equipped with a ZigBee transceiver for communication with the central unit. In case of ploughing, first, few robots are sent out for scouting the area. Then the next group of robots is sent to plough the field based on the obtained scouting data in the central unit. In the presented work, the task allocation is carried out based on the initial position of the robots as well as the received data from the central unit. The presented system is robust and scalable, but the individuals are sharing their information by a central unit. This makes the system vulnerable to the loss of the central unit.

Li et al. (2015) presents another central-based cooperative model for a heterogeneous team of ground robots applied in citrus harvesting activity in a known environment. In this example, a robot team is divided into multiple smaller groups with local leaders, and each group is sent to different tree locations. At each location, the robots in each team are divided into two groups: one to shake the tree, and one to collect the fruits. At the beginning of the task, when the initial signal is received by any individual in the team, the central unit, which knows the position of all robots and the tasks, selects an appropriate formation from a predefined collection and chooses the leader in the team based on their positions in the selected formation. Once the leader is determined by the central unit, the position of the followers also are determined based on the selected formation. Again, if the central unit fails the entire unit will fail.

Noguchi et al. (2004) developed a master-slave team of two robots that constantly communicate over Wi-Fi for distance commanding and distance adjustment. The leader robot makes the decisions and transmits appropriate commands to the fol-

lower robot. The non-leader robot follows the leader and sends its status in terms of current location back to the leader at the frequency of 2Hz. The cooperative execution is carried out by two behaviour-based algorithms: FOLLOW and GOTO. In GOTO algorithm, the leader requests the follower to go to a specific location in the field, and in FOLLOW algorithm, the leader wants the non-leader unit to simply follow the leader at a specific distance with a specific angle.

Similarly, Zhang and Noguchi (2016) presents a cooperation mechanism for a team of two driverless tractors for cooperative navigation in an open field. In this approach, a robot is either client or server. The server robot receives the information from the client robot via Bluetooth messages, and it performs series of calculations to obtain the correct velocity and path for itself and the client robot. It is worth noting that the paths are predetermined and they are given to both robots.

Roldan et al. (2016) developed a heterogeneous team of two robots for the purpose of greenhouse monitoring. The team consists of an Unmanned Ground Vehicle (UGV) and an Unmanned Arial Vehicle (UAV). the UGV carries the UAV on a platform while it develops its tasks, and when it is required, the UAV takes-off, performs some tasks and lands on the UGV (Roldan et al., 2016). To increase the robustness, a charging pad on the UGV is mounted so the UAV can be recharged while it is being carried, also, there are charging locations in the greenhouse for refuelling the UGV. The UAV carries the required monitoring sensors (including humidity, CO2, and Temperature), and the team travels to particular locations to measure the required variables.

Arguenon et al. in 2006 developed a multi agent system in simulation for transporting grapes in vineyard. The proposed system is for a heterogeneous team of robots. There are two types of robots in the team: large transportation robots and small

transportation robots. Large transportation robots have three different behaviours: move to one of the transport robots, move to the transportation centre, or wait for instruction. Small transportation robots also have three rules: move to one of the transport robots, move to one of the harvesting robots, or wait for instructions. Each robot, regardless of their size, must have some displacement strategies or behaviours: ReachTarget which defines the desired location for the robot, Perception which is a procedure consisted of obtaining some information about the neighborhood of the robot from the different sensors available and Avoid whenever an obstacle is detected and the desired location of the robot can be called into question, as for example when it cannot move. Arguenon (2006) utilised a centralised system to provide a global view of the farm to transportation robots over Wi-Fi by sending specific messages whenever it is requested. Each message consists of four main components: a state which refers to the context of the message, recipient which corresponds to the ID of the robot that should receive the message, transmitter which corresponds to an ID of the robot that sends the message, and location which refers to the location of the transmitting robot.

In a similar application, Kong et al. (2006) presents a cooperative mechanism by which a team of autonomous and homogeneous lawn mowers processes a field. The robots in this approach are assumed to have communication capabilities. At the beginning, the field is decomposed into smaller pieces, and each is allocated statically to a robot. As robots are processing the allocated section, they regularly share their information to optimise their performance and determine where to process next using customised task selection protocols.

Saska et al. (Saska et al.) presents a homogeneous team of robots for a snowplough application at the airport. The system aimed to control the robots in the sweeping

and moving modes. In the sweeping mode, the snowplough formations are capable to completely cover the surface of runways by following their axes. The moving mode is important for the autonomous design of manoeuvres required for displacement of the formation into the position of the next sweeping task. Besides, the turning on the spot in a scattered environment is enabled for large formations in the moving mode. This is an important skill of the system that enables to turn the sweeping formations at the end of blind runways or in a case of runways blocking. The demonstrated approach utilises the leader-follower method in which non-leader robots track the leaders trajectory in a predefined spacing as if they interact with each other using the environment, and robots maintain their relative distance to the leader in a curvilinear coordinate system.

2.6.3 The Problem of Localisation in Agricultural Robotics

In autonomous open-field farming in which agricultural tasks such as ploughing and spraying need to be performed in a large area, the need for navigation and localisation is prominent. Without localisation, mobile robots will lose their orientation and global position in the farm which result in the failure of the operation. Numerous researches have been conducted and as a result, different localisation systems in the farm have been developed and successfully field tested.

Generally, localisation and position measurement can be divided in two main groups: relative (also known as dead-reckoning) and absolute (also known as reference-based) position measurements. In relative localisation, positions can be measured relatively using Odometry and inertial navigation sensors such as Gyroscope and Accelerometer (Borenstein et al., 1997). However, relative approaches (standalone) do not provide sufficient accuracy for agricultural purposes. For instance, odometry, which

is inexpensive with good short-term accuracy with the high sampling rate, consists of systematic and un-systematic errors (Borenstein and Feng, 1996). Systematic errors are resulted from kinematic imperfections of the robot and are measurable using UMBmark test. The source of unsystematic errors is the floor that the robot is moving on, so measuring such error is not useful especially in the farm where the tracks and roads are bumpier and slippage and sticking are more probable. On the other hand, inertial-based navigation systems which do not need any external references are subject to drift over time (Borenstein et al., 1997).

Alternatively, robots position can be obtained using reference-based techniques. According to Borenstein et al. (1997), these techniques can be categorized as follows: Magnetic Compasses, Active Beacons, Global Positioning Systems, Landmark Navigation, and Model Matching.

Global Positioning System: Nowadays GPS is widely being used to obtain the absolute position of the robots in an open field. Researchers at Stanford University successfully guided a John Deere 7800 tractor on prescribed straight row courses with headland turns (Bell, 2000). Although the results were promising, the accuracy wasnt enough for precision farming. Instead, real-time kinematic version of GPS (or GPS-RTK) brought enough accuracy for the localisation problem in agriculture. Localisation systems demonstrated by (Stoll and Kutzbach, 2000) and (Thuilot et al., 2001) capable of navigating a vineyard or orchard using GPS-RTK are the examples of such localisation system. This navigational technique is suitable for open fields due to the fact that GPS signals could not be interfered by RF signals in the field.

Navigational Landmark-based Localisation system: The lack of GPS availability due to environmental conditions such as large canopies need for prior surveying of the area, and unreliable connectivity in certain areas compels to search

for GPS-free localisation techniques. Marden and Whitty (2014) introduced a GPS-free approach for navigating in a vineyard. The system utilizes LiDAR sensor to extract point clouds from the environment and perform simultaneous localisation and mapping (SLAM). In this work, Localisation and mapping is performed by extracting features from the rows in the vineyard in an EKF-SLAM framework. Then, RANSAC (RANdom Sample Consensus) is used to extract lines from the 2D LiDAR laser scan data and parametrised in polar coordinates. Then the navigation is achieved by the specific control law. The system has been tested intensively in simulation only. The results of the system promise 2.5 meters accuracy which is enough for the field of application. Nonetheless, the localisation accuracy slowly drifts over time. However, in open field area in which landmarks are scarce, this technique cannot be utilised.

Vision-Based Localisation System: In another GPS-free approach, tractors are able to localise themselves in the farm using artificial landmarks. Imou et al. (2009) describes a system in which an Omni-directional camera is used to trilaterate the position of a robot in the farm. The key advantage of this approach is the robot has a panoramic view of its environment, which makes it possible to create features that are variant to the robots orientation, yet each tractor requires direct line of sight with all available landmarks, so this approach is not suitable for multi robot system as one robot could block each other's view.

RFID Localisation System: The recent advances in RFID (Radio Frequency IDentification) have made this particular technology more useful for agricultural industry. The most important characteristics of RFID tags is that they do not require line of sight to be read, hence they are suitable in the crowded environment or in multi robot scenario. Moreover, having longer range readability (even more

than 100 meters), fast reading rate (up to 100 tags per second), and data storing capabilities (up to 4KB in passive mode and up to 1 MB in active mode) will make them even more suitable for multi robot and agricultural applications (Ruiz-Garcia and Lunadei, 2011).

RFID can be used in food traceability, animal identification and tracking, livestock application, and precision agriculture. In precision farming, RFID tags can be used for localisation purposes, however, the accuracy of the localisation results depends completely on the type of RFID that is being used. Generally, there are two types of RFID tags: Active and Passive. A passive RFID tag comprises a micro-circuit and an antenna, and they are active whenever they are within the range of RFID readers. Active RFID tag, however, is integrated with an external power source which gives the active tag the capability to transmit information about itself at the greater range either by constantly transmitting information or whenever it is prompted. One great advantage that active RFID tags could bring to the system is that they require very low signal strength from the reader to initiate their operation. This means that they can operate in the noisy environment and/or long range. Moreover, their range could go to 100 meters which make them suitable for trilateration based localisation for outdoor applications. It is worth mentioning that trilateration is a method to determine the position of an object based on simultaneous range measurements from three reference nodes at a known location using only the magnitude of the received signal also known as RSSI (Received Signal Strength Indicator) (Chung and Lau, 2007).

Yet, different researches have utilised passive RFID tags in localisation application. In (Choi et al., 2009), a system is developed to use the properties of PASSIVE UHF RFID such as RSSI (Received Signal Strength Indicator) to localise a robot in an

outdoor environment. The result was not satisfactory as the RSSI value can be easily changed by the environment. Active RFID tags, on the other hand, provides more accurate and consistent localisation information as demonstrated in various researches. In (Huang et al., 2006), a cost-effective probabilistic approach is simulated via MATLAB with errors less than 7 meters. Chawla and Robins (2011) proposed an accurate, scalable and reliable RFID-based approach for localising objects by deploying four active RFID tags in the environment and used RSSI and intersecting circle technique to estimate the location. They also calibrated the tags to overcome the problem raised by Choi et al..

Hybrid Approaches: Despite shortcomings of relative positioning sensors (odometry, Accelerometer, and Gyroscope), results of infusion of dead reckoning solutions with other techniques (including low-cost GPS receivers) are to some extend promising. In (Oksanen et al., 2005), a tractor guiding system is developed by infusing IMU (Inertial Motion Unit) data with coordinates obtained from a low-cost GPS receiver via first order Kalman filter. It was concluded that the system is reliable for short distances as short time noise type errors in positions can be eliminated, but longtime bias type error is impossible to eliminate. In another example, coordination obtained from a low-cost global positioning system is infused with low cost inertial sensors and a technique for vision-based row tracking (English et al., 2013). Vehicle roll and pitch is estimated with the infusion of visual horizon detection and IMU data which are combined with a simple Kalman filter. As a result, the positioning system is capable of handling long correction GPS signal dropouts.

In the current research, we are not considering the problem of localisation in the team since the current results from the related works can successfully provide the required information. The currently available localisation systems for agricultural

applications can provide up to centimetre precision and they are commercially available. One example of such navigation system is AutoDrive developed by ATC (Autonomous Tractor Corporation). AutoDrive is an autonomous navigation system which does not suffer from dead spot or no reception like the GPS-based system and it is not affected by sun-spots. One advantage of AutoDrive is its integrability to any autonomous tractors. John Deere, an agricultural machinery company, has developed an autonomous tractor which utilises AutoDrive. The current direction of research in this field is aimed to reduce the cost of the proposed approaches while maintaining high accuracy up to few centimetres.

2.7 Conclusion

In this chapter, we reviewed the main problems, which is occur in every cooperative team of robots, possible directions to resolve them, and analysis of the related solutions. Our investigation suggests that the current trend in multi robot systems in the agricultural application domain is to resolve the cooperation solutions via explicit forms of communication. This is insufficient and unreliable because depending on explicit forms of communication makes the system vulnerable to the loss of the communicating signal, limits the area of coverage, limits team flexibility to the dynamic changes that could occur during execution, and increases the computational complexity of the system. Instead, the current research aims to resolve the cooperation related problems via implicit forms of communication. Specifically, it aims at developing a fully autonomous self-organising robotic team that does not depend on the number of participating robots (scalable), does not require neither central unit nor explicit negotiation, is affected by loss of a participating robot (robust), and can be further improved or integrated with different teams (heterogeneous and flexible). To the best of our knowledge, such system does not exist in the community of agricultural robotics and it is the first of its kind.

In the following chapters, we look into three different tasks: ploughing, spraying, and harvesting. As mentioned, these tasks provide different constraints to the team of robots, thus robots have to behave differently.

CHAPTER 3

COOPERATIVE PLOUGHING: DESIGN AND IMPLEMENTATION

In this chapter, ploughing in a team of robots is discussed. In 3.1 the task of ploughing is analysed, and the design considerations and requirements are described in 3.2. Potential issues including inter robot interaction model between robots for task allocation using local data acquired from 2D-camera (3.3), congestion avoidance (3.5) using pre-defined navigational conventions and flee-opposite algorithm, and collision avoidance (3.6) using artificial potential functions are described in words along with appropriate flow charts and the implemented C++ codes in appendix sections. Two team level cooperative algorithms (FIFO and LIFO) for ploughing are described in two different sections: the main ploughing (3.7), and the furrow transitioning (3.8). The proposed approaches are analysed and an optimised approach (self-organised) is described, and compared with other approaches in (3.9).

The points of this chapter have been published in the following:
Janani, A., Alboul, L. and Penders, J., 2016, May. Multi-agent cooperative area coverage: case study ploughing. In Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (pp. 1397-1398). International Foundation for Autonomous Agents and Multiagent Systems.

3.1 Ploughing Analysis

Ploughing is part of the seedbed preparation process, and it is carried out by dragging a ploughing mouldboard across the field. The ploughing mouldboard digs deep into the soil and disperses the soil in one direction. Ploughing removes soil compaction, buries the weed, and brings up the nutrient materials to the surface.

Ploughing creates a two-part pattern: (1) A narrow trench referred to as *furrow*, and (2) a hill-top soil which is called *ridge*. The dimensions of the furrows depend

on various environmental factors (including soil type, the frequency of rain, and etc) and the type of crop that will be harvested. However, the distance between two consecutive furrows ranges between 25 cm and 50 cm.

Figure 3.1: A farmer is ploughing a field using tractor equipped with conventional mouldboards (Fao, 1997).

3.1.1 Ploughing Patterns

Ploughing determines the amount of crop that can be harvested. There are two patterns by which a field can be ploughed: **straight ploughing** (Figure 3.2(a)), and **zigzag ploughing** (Figure 3.2(b)).

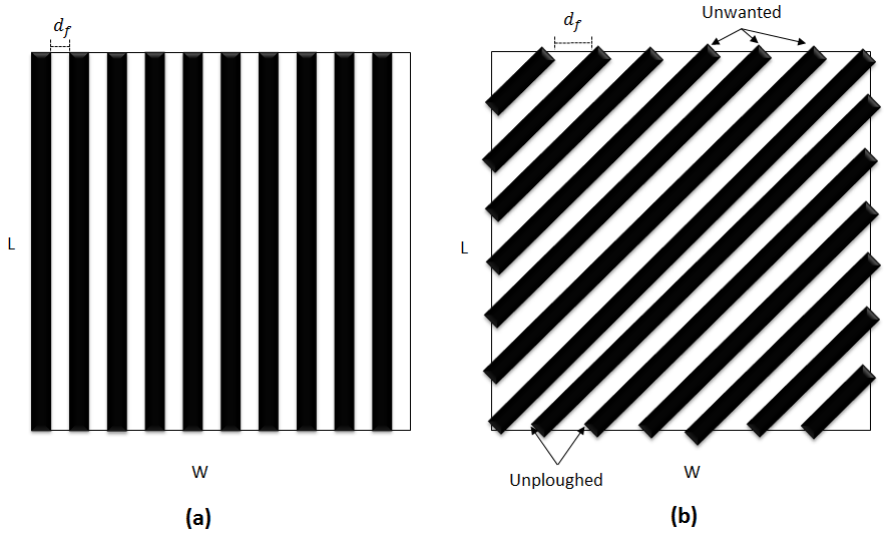


Figure 3.2: Ploughing patterns: (a) Straight ploughing, (b) Zigzag ploughing.

Both approaches provide the same area for ploughing given the following:

$$L \times W = w_f \sum_{1}^{K} l_{f_i} \tag{3.1}$$

In here, the length and width of the field are represented by L and W respectively, w_f is the width of a furrow, K is the number of furrows ploughed in a field, and l_f is the length of a furrow.

However, between the two ploughing patterns, straight ploughing distributes the same cultivable (meaning farmable) area with the lower number of furrows which also means less turning at the end of each round of ploughing.

Every time a furrow is completed, the ploughing unit has to travel to the beginning of the next furrow. This stage is referred to as *furrow transitioning* in which the ploughing unit uses the top side or bottom side of the field to travel to the next furrow (see Figure 3.3). The area that is used for furrow transitioning is called *headland*. Headlands are unprocessed areas that should be kept as small as possible since they limit the length of cultivable area (h_1 and h_2 in Figure 3.5 and 3.6).

Figure 3.3: Ploughing units are performing furrow transitioning (Estate, 2015)

This allows future field processing (including seeding, spraying, harvesting, and etc) to be executed faster. Furthermore, since in straight ploughing the furrows are

created homogeneously, the field can be processed simultaneously. Whereas, with zigzag ploughing each furrow has to be processed separately. Therefore, straight ploughing is the most popular and commercially endorsed ploughing pattern. Thus in this research, only ploughing in straight lines is considered.

3.1.2 Ploughing Restrictions

When a furrow is created, a ridge is also created. During ploughing, the ploughing unit uses the previously created furrow to navigate while the soil removed to create a furrow is turned up and dumped in the previous furrow (see Figure 3.1). In order to keep this pattern, three rules have to be followed: (i) a location can be ploughed if and only if the previous location is ploughed, (ii) furrows can be created only once, and (iii) furrows cannot be created simultaneously. These restrictions make ploughing a sequential and dependent task.

3.1.3 Ploughing Mouldboards and Furrow Transitioning

Besides the restrictions caused by the nature of ploughing, the type of ploughs constrains the task execution even more. There are various types of ploughs (e.g. Chisel, Ridge, Disk, and etc), but plough mouldboards are the popular and widely used ploughs. Plough mouldboards can be classified into two categories: (1) *conventional mouldboards* by which the soil is dispersed in one fixed direction, and (2) *reversible mouldboards* by which the dispersion of the soil can be set in two opposite directions. Traditionally, a plough would have single mouldboard by which only a single furrow could be created at a time. However, nowadays ploughs have multiple mouldboards and few furrows can be created at a time. This reduces the ploughing time, but the mouldboard weight increases significantly, and in return, they require stronger and bigger tractors to drag them across the field.

Figure 3.4(a) and (b) demonstrates 4-plough conventional and reversible mouldboards respectively.

(a) (b)

Figure 3.4: Plough mouldboards: (a) 4-plough conventional mouldboard, (b) 4-plough reversible mouldboard. (Agriavis, 2015)

A ploughing unit is capable of creating a limited number of furrows at a time, so it has to repeat ploughing until the field is completely ploughed: *maximum number of furrows are created*. Maximum number of furrows in a field can be calculated as follows:

$$K = \lfloor \frac{W}{d_f} \rfloor \tag{3.2}$$

In here, W is the width of the field, d_f is the width of a furrow, and $\lfloor \frac{W}{d_f} \rfloor$ represents the largest previous integer.

The attached mouldboards determine where to plough next. With conventional mouldboards, ploughing is carried out in loops, and either starts from one side of the field, and ends in the middle (Figure 3.5 (a)), or starts in the middle and ends on one side of the field (Figure 3.5 (b)). Whereas with reversible mouldboards, ploughing starts in one side of the field and ends on the other side of the field(see Figure 3.6). With a reversible mouldboard, furrows are created in a consistent pattern (i.e. a ridge will always accompany a furrow). However, with conventional mouldboard, the created pattern on the field is inconsistent, meaning that in one

half of the field the soil is dispersed into one direction and on the other half the soil is dispersed on the opposite direction. Consequently, this will result in either "Closing Furrow" (two consecutive furrows) or "Opening Crown" (two consecutive ridges) in the middle of the field.

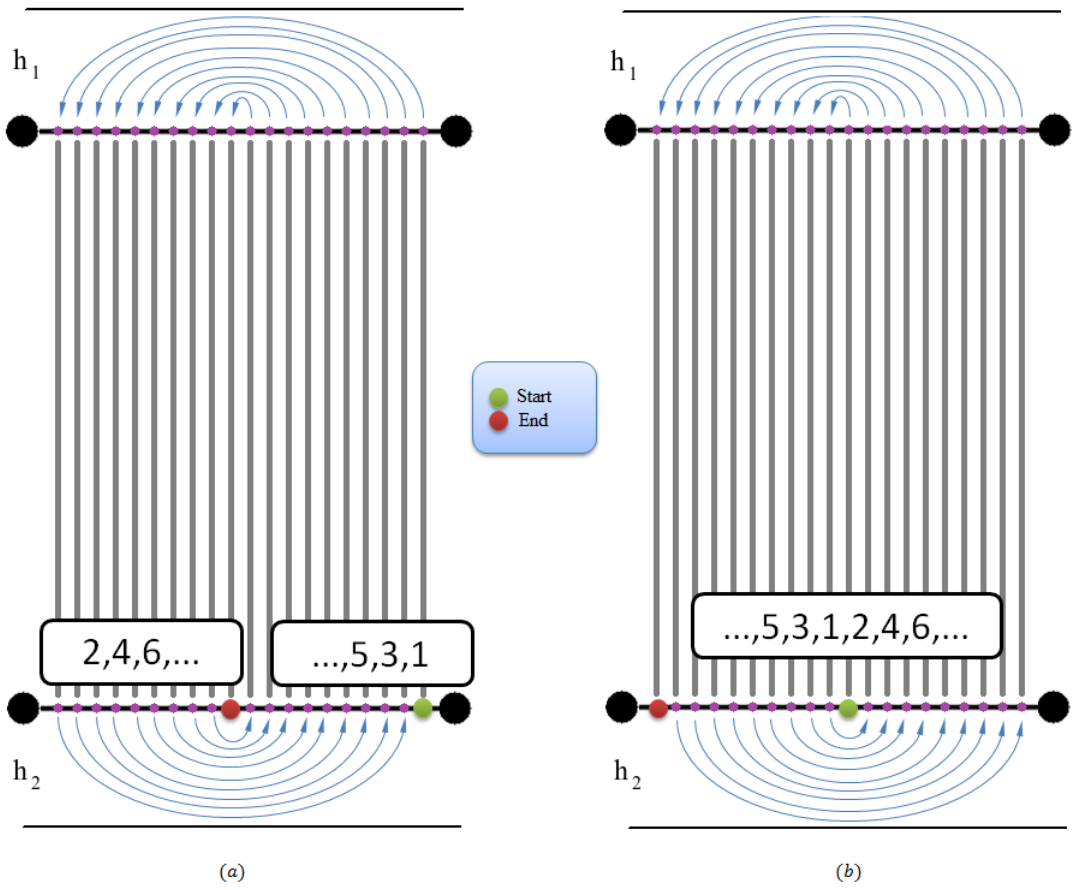


Figure 3.5: Pattern of ploughing with conventional mouldboard: (a) ploughing starts on one side and ends in the middle, (b) ploughing starts in the middle and ends on one side.

3.1.4 Ploughing Cost

The task of ploughing can be re-expressed as a navigational task in which a ploughing unit has to travel between series of coordinates. While navigating, the ploughing unit is either ploughing or furrow transitioning. With this, a ploughing cost function can be expressed in terms of the travelled distance as follows:

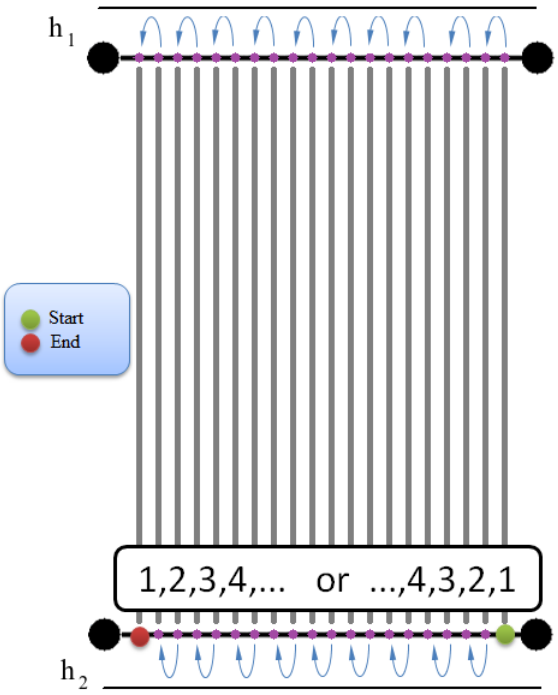


Figure 3.6: Pattern of ploughing with a reversible mouldboard: ploughing starts from one side and ends on the other side.

$$PC = dist_{pl} + dist_{ft} \tag{3.3}$$

Where, PC is the ploughing cost, $dist_{pl}$ is the required travelled distance during ploughing, and $dist_{ft}$ is the required travelled distance for furrow transitioning.

If only one furrow can be ploughed at a time, regardless of the attached plough mouldboard, the required travelled distance during ploughing of K number of furrows with l_f as the length of a furrow can be obtained as follows:

$$D_{pl} = K.l_f \tag{3.4}$$

However, the type of the attached ploughing mouldboard determines how far the ploughing unit is required to travel during furrow transitioning period. With conventional mouldboard, the required travelling distance during furrow transitioning

is determined as follows:

$$D_{ft} = \frac{K}{2}(W - d_f) \tag{3.5}$$

Whereas, with a reversible mouldboard the required distance for furrow transitioning is:

$$D_{ft} = (K - 1).d_f \tag{3.6}$$

Where, D_{ft} is the required travelling distance for furrow transitioning, K is the number of furrows, and d_f is the distance between two adjacent furrows.

Comparing 3.5 and 3.6, it can be concluded that ploughing with a reversible mouldboard is more cost effective since the travelling distance required during furrow transitioning is reduced significantly.

3.2 Design Requirements and Considerations

Section 3.1 considers generic ploughing with a single unit (equipped with a single plough mouldboard). In the current section, ploughing by a team of robots is discussed. But before we proceed, the term *cooperation* within the context of cooperative ploughing has to be defined.

Definition 3.1 *A field is cooperatively ploughed if the sum of furrows ploughed by each participating individual is equal to the total number of furrows in the field (K):*

$$K = \sum_{i=1}^{n} k_i \tag{3.7}$$

To perform ploughing in a team within the mentioned restrictions, an order has to emerge among the participating robots. However, as it is demonstrated in Figure 3.5 and 3.6, depending on the attached mouldboard, the furrows are to be ploughed

in a particular order. This demands different ordering among robots.

In cooperative ploughing, there are two main questions to address: (1) *How to identify the first furrow to plough?* and (2) *Where to plough next after completing the first furrow?*

There are various approaches to divide and distribute furrows among robots. Clearly, to assure that furrows are ploughed consecutively, furrows cannot be statically distributed among robots. This is because robots may be scattered initially around the field, and attend their assigned furrow locations at different instances of time. This will violate the ploughing order discussed before. Therefore, the distribution of furrows has to be carried out in real-time.

As mentioned in Chapter 2, a popular trend for real-time and dynamic task allocation and coordination in multi robot system is to utilise a central unit. The central unit could be a dedicated stationary unit or one of the participating robots. Clearly, robots are required to communicate with the central unit via explicit forms of communication to exchange information. One major problem with this approach is the susceptibility to the loss of contact with the central unit. The success of the system depends on the state of the central unit. If the central unit fails, no furrows will be ploughed. Furthermore, robots have to be within signal coverage distance at the time of task allocation, otherwise, they cannot contribute to the task. This normally occurs when robots are scattered around a large field. This approach may be suitable for small size fields, but for large size fields (bigger than the signal coverage distance), which is normally the case, the central unit fails to communicate with other robots. Therefore, cooperative ploughing task allocation should not rely on explicit forms of communication.

So, what are the characteristics of the appropriate approach for task allocation and

coordination in cooperative ploughing? The proposed approach has to be scalable (independent of the number of participating robots), distributed (no central coordinator: robots make their own decisions based on the collected information), self-organised (the cooperative behaviour should emerge as a result of their local interaction and not based on negotiation), independent of any initial position (e.g. robots should not be at particular location) so that the same team can contribute to other farming tasks (e.g. spraying, or harvesting).

3.3 Interaction Model

For task allocation and coordination, robots have to interact with each other. As mentioned, robots have to rely on their local information to perform task allocation, collision-free navigation, and congestion clearance. This includes detecting and differentiating other robots from the rest of the objects in the environment and extracting marks or hints that other robots left in the environment as a result of their execution. But *what kind of information can be extracted from ploughing? and how should they be interpreted?*. In brief, if a robot can determine whether a location is ploughed or not, then it can find out by itself where it is required to plough next.

3.3.1 Furrow Detection

As mentioned before, as a result of ploughing two-part pattern is created: furrows and ridges. Participating individuals could use previously created furrows as an indication to determine whether a location is ploughed or it is required to be ploughed. Previous works suggest that it is possible to analyse the soil to detect any existing pattern. One of the widely common approaches for furrow detection is via vision sensors.

In various single robotic agricultural works, furrows and ridges are used to guide unmanned vehicles within the field. Vision based furrow and ridge detection has been applied in real world application including cauliflower (Marchant and Brivot, 1995), cotton (Billingsley and Schoenfisch, 1997; Slaughter et al., 1996), tomato and lettuce (Slaughter et al., 1996), sugar beet (Marchant, 1996),and cereal (Hague and Tillett, 2001; Sgaard and Olsen, 2003).

In the early works, near-infrared images were popular to extract the desired features. In Tillett et al. (2002), a guidance system for navigation between sugar beet rows is described. Tillett et al. utilised band-pass filter to extract lateral crop row location. In another example, Astrand and Baerveldt (2002) applied grayscale Hough transform on images provided by near-infrared camera to detect sugar beet rows.

In recent years, by the development of high-quality inexpensive cameras, vision-based furrow and ridge detection become even more popular since image processing can be carried out simpler, quicker, and more accurately. Tang et al. (2013) presents a real-time row detection method for an autonomous weeding robot to robust detection of wheat row based on pixel histogram. This method involves two steps segmentation of grey scale image and wheat row detection based on after segmentation. In Zhang and F. (2013), color images are transformed into saturation images and binary images sequentially. Using Radon transform and the position of the robot, the furrows lines are located. In Tian et al. (2014), navigation paths are extracted by simply applying image segmentation and morphological filters using MATLAB software.

3.3.2 Vision Based Furrow Detection

In order to examine the feasibility of furrow detection, a vision based approach is developed and implemented. Analysing the close-up images of freshly ploughed field reveals that the main difference between a ploughed and unploughed field is the pixel intensity of the soil. The ploughed area contains darker pixels, compared to the unploughed area of the field.

In the implemented approach, an inexpensive camera is bundled in the front of the robot. At each ploughing location, the robot stops and analyse the ploughing location for the existence of a furrow. The furrow detection procedure is as follows: First, the captured RGB-image is converted to gray-scale colour space. Then, a binary image is created with the threshold value obtained from the histogram analysis of the grayscale image. Next, the image is inverted so that the area of interest is represented as white spots. To remove the noise from the image, morphological filters are applied on the image. The image is eroded using a rectangular sized matrix (10 X 3), and then it is dilated with the same matrix. The matrix is chosen in this shape because it is similar to the shape of a furrow from the perspective of the robot. The proposed vision system is applied to various images taken from the field with different background light condition (see Figure 3.7). Although the results obtained from these analyses are not sufficient for guidance purposes within the field, it is enough to determine if a location is ploughed or not.

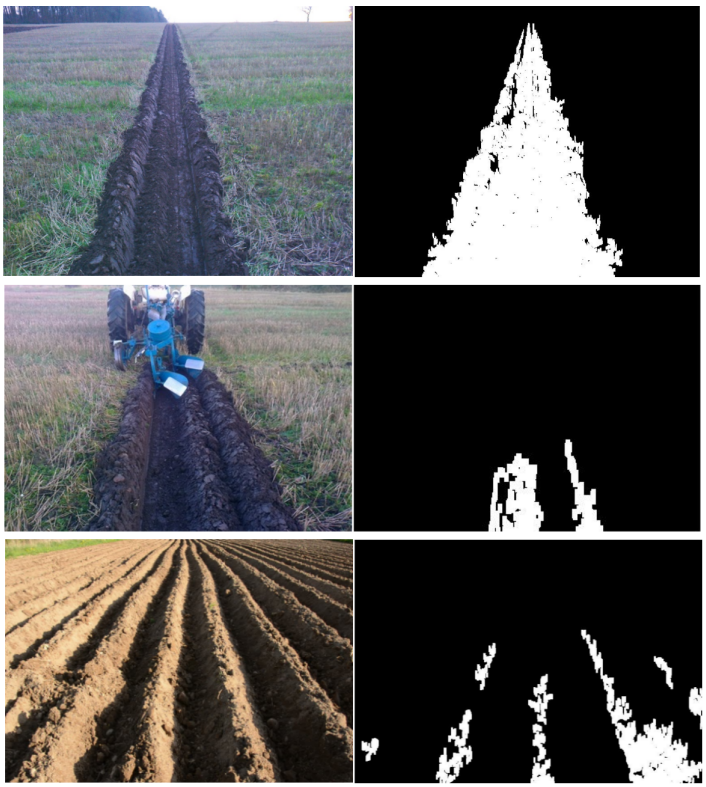


Figure 3.7: The vision based furrow detection algorithm can successfully separate furrows from the background environment under various light conditions.

3.3.3 Accuracy of the Vision Based Furrow Detection

Using the online resources, various ploughing picture similar to the ones displayed in Figure 3.7 were collected and passed through the implemented vision algorithms. As a result the accuracy of 70.35 percent was achieved (i.e. from 200 testing pictures, furrows in 141 pictures were successfully detected). This is because of the nature of the approach. As mentioned, the approach first finds the histogram of pixel intensities, then looks for local maxima with the neighbour gradients higher than a threshold in the resulted histogram as they are considered to be furrows. Now, if the furrow pixels' intensity are not dark enough, the peak of the local maxima drops below the threshold and it cannot be identified as a furrow. One main factor in this phenomenon is the depth of the ploughed furrow. If the ploughed furrow is not deep

enough the required properties cannot be achieved and the detection will fail. In other words, the deeper the furrows, the more accurate the detection. This is the key reason that the reviewed approaches are successful. For instance, the reviewed methods including (Marchant and Brivot, 1995), (Billingsley and Schoenfisch, 1997), (Slaughter et al., 1996), (Marchant, 1996), (Hague and Tillett, 2001), (Sgaard and Olsen, 2003) in which the furrow detection is used as a method for guiding the autonomous units, unanimously claim errors less than 20mm and less than 2.0 degrees in heading. In all mentioned approaches, there is a significant change in pixel intensity between the crop and the surrounding soil, therefore it is easily detectable with traditional image processing techniques including the method presented in this thesis (i.e. Histogram Analysis).

Due to this shortcoming, more reliable detection methods should be applied. One available solution is to use machine learning approaches and in particular deep learning. By providing negative and positive examples, a model can be trained (i.e. a convolution neural network in the deep learning) with which high accuracy in classification and detection can be achieved. The accuracy depends on the number of provided positive and negative examples, how different the examples are from one another, and the type of convolution neural network used.

However, one major drawback of deep learning approaches is how expensive the process is computationally. Deep learning approaches requires extensive amount of computational resources. It is so demanding that, for optimization, the models are normally run on NVIDIA graphic cards, and this will increase the cost of the system.

3.4 Points of Failure

Distributed team of robots operating in a shared environment raises certain concerns with which failure in the team becomes inevitable and therefore they have to be addressed. In here, these problems are briefly described.

In general, the failure could be happening either outside of the field (during task allocation) or during the actual individual task execution (e.g. ploughing, or spraying). The latter, in case of sequential tasks such as ploughing, requires further planning and it is outside of the scope of this research. It is recommended as one future work to this project. Therefore, only the failures outside of the field is considered.

Let's consider the scenario in which a team of robots are distributed around a large field randomly. When the task is started, all available robots navigate toward the field entrance point. At this stage, any collision with static and dynamic obstacles in the environment will result in the elimination of that unit from the team.

In addition, as all robots are targeting one shared location in space, congestion due to acquiring the spacial target simultaneously will prevent the robot to proceed the field and will result in the team failure.

In the following sections, more details for the mentioned issues are provided and appropriate solutions are listed.

3.5 Congestion Clearance

Since robots operate in a shared environment, the occurrence of congestion in the field becomes inevitable. The congestion has to be resolved or avoided otherwise it will result in the failure of the given task. In this research, congestion results from two main sources: congestion as a result of spatial resource conflict, and congestion as a result of a collision with dynamic and static obstacles in the environment. In

this section, the solutions for overcoming both congestions are described.

3.5.1 Spatial Resource Conflict

Robots have to approach and analyse ploughing locations one by one to determine the state of the task (i.e. the location which is required to be ploughed next). Since robots are initially scattered around the field, they approach the ploughing locations from various directions. Clearly, robots should not pass through the field as they are unaware of the state of the task (i.e. robots do not know how much of the field is ploughed). Figure 3.8 illustrates the permitted directions of approach to the first ploughing location.

When approaching the first ploughing location, two or more robots could reach the targeted location at the same time from different directions (Figure 3.9 demonstrates the possible direction of approaches to the first ploughing location). The question that has to be addressed is *which robot will take the priority in movement?* (i.e. spatial resource conflict).

3.5.2 Proposed Solution to Spatial Resource Conflict

To resolve the probable resource conflict, a robot has to decide to either give priority to another robot or to ignore the existence of others and takes the path. This decision has to be made only on the basis of the local information of the robot.

For this, first the field of view of the robot is divided into three regions: Left Region (**LR**), Front Region (**FR**), and Right Region (**RR**) (see Figure 3.10 (a)). Then, using the following conventions, a robot can decide whether to take priority or not.

- **Left Convention**: If another robot is detected on the left region (LR), ignore and take the priority.

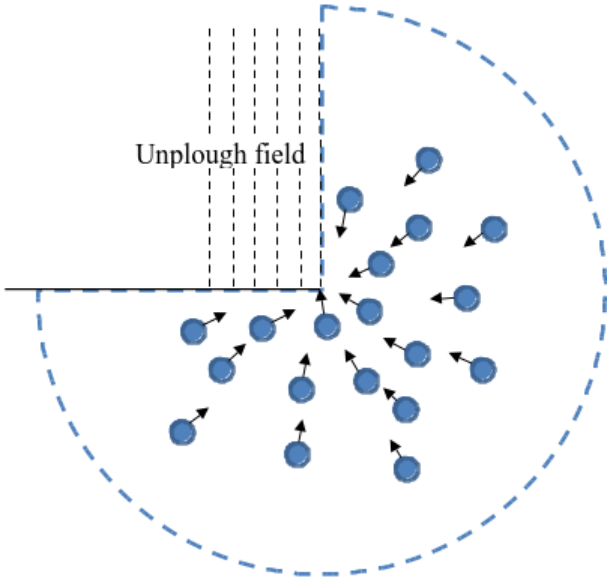


Figure 3.8: Robots approach the ploughing locations from different directions. Robots should not pass through the field to examine the ploughing locations as the state of the task is unknown.

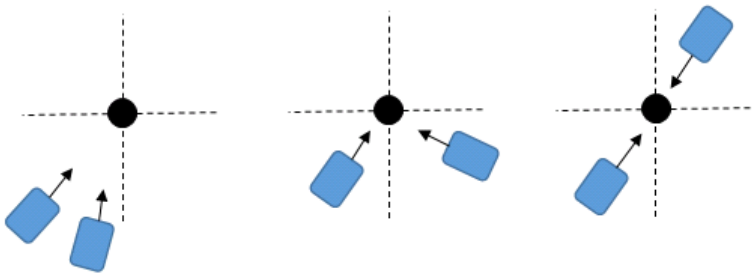


Figure 3.9: different direction congestion: robots approaching the same location from different side of the target.

- **Right and Front Convention**: If a robot is detected in front (FR) or on the right region (RR), wait (i.e. stop with zero velocity) and give priority to the detected robot.

However, sensors are subject to noise, and they cover only a limited range. This, sometimes, leads to not detecting robots in the regions of interest. This is common when robots are very close to the shared location. To rectify this, a threshold distance is defined on the left region. With this, the detected robots will be considered

on the left region if the distance becomes less than a certain threshold distance. The region of interest in the field of view of a robot is demonstrated in Figure 3.10 (b).

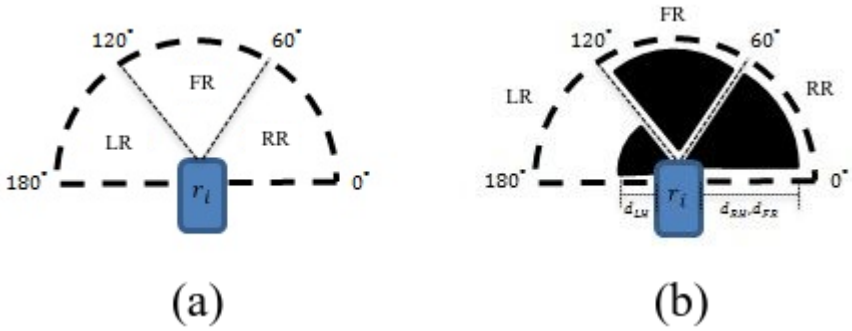


Figure 3.10: Robots are assumed to have a limited field of view. (a) The field of view of the robot is divided into three regions: LR (Left Region), FR (Front Region), and RR (Right Region). (b) Applied conventions on the regions.

The conventions are only effective when robots are approaching from the same direction. However, it is completely ineffective when robots are approaching from opposite directions since both robots stop for the path to be cleared.

There are two reasons why congestion still occurs: (I) The defined conventions make the robots to wait for another robot to take initiatives to clear the path. (II) Robots behave homogeneously (i.e. robots behave the same upon encountering another robot in the aforementioned regions).

In order to break the homogeneity in robots' behaviour, instead of waiting at the time of detection, robots move reversely with random velocity until the triggered region is cleared (i.e. the robot is no longer in sight). This idea is inspired by *flee-opposite* which is a straightforward obstacle avoidance method described in (Penders, 1999). The simulation video for the proposed solution is available in (Janani, 2018c) [1].

[1]Congestion Avoidance Video Link:

3.5.3 Collision Avoidance

Besides resource conflicts, collisions with dynamic and stationary obstacles in the environment also result in congestion. To avoid collisions, in this research, Artificial Potential Field (APF), first introduced by (Khatib, 1986), is implemented on each robot. According to Arkin (1998), APF is the most common continuous response method approach in which only overview of the surroundings of the robot is considered. The APF is fast, computationally inexpensive, and easy to implement (Wu et al., 2016b).

The Artificial Potential Field is inspired by the particles interaction model. Particles with the same polarity repel each other, whereas particles with the opposite polarity attract one another (see Figure 3.11). As a result, the potential function is the sum of the corresponding attraction and repulsions fields and it can be expressed as follows:

$$U(q) = U_{att}(q) + \sum U_{rep}(q) \tag{3.8}$$

In robotic terminology, a robot is considered to be in the opposite polarity with the goal, and in the same polarity with other obstacles in the environment. Consequently, the robot is always attracted to the goal location while moving away from the surrounding obstacles.

A popular and widely used attractive potential function is quadratic well (Park et al., 2001). It is simple and provides a linear control law with constant gain. The quadratic well attraction field is expressed as follows:

$$U_{att}(q) = \frac{1}{2}\omega_a |q - q_d|^2 \tag{3.9}$$

Where ω_a is the attraction gain, q is the position vector of the goal, and q_d is the

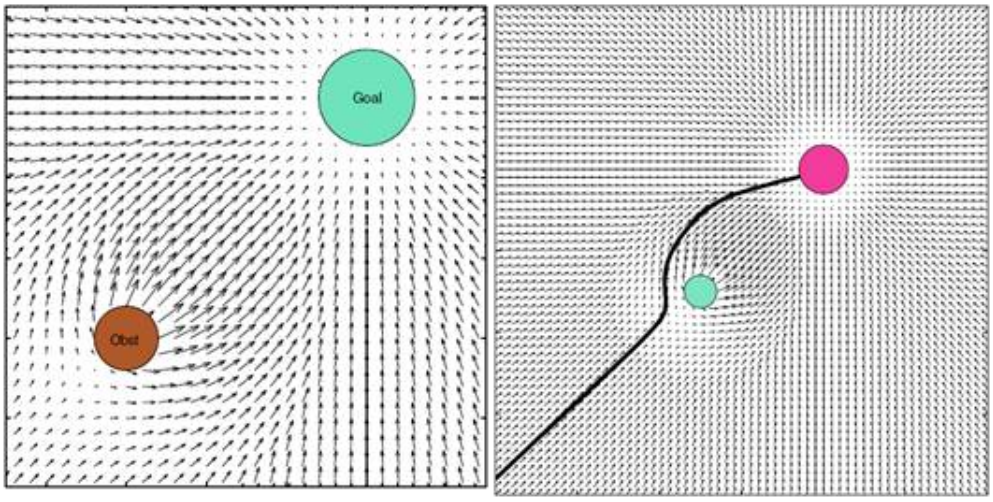


Figure 3.11: In Artificial Potential Field, the goal, which is in the opposite polarity of the robot, attracts the robot and the obstacles, which are in the same polarity as the robot, repel (Safadi, 2007).

position vector of the robot. The attraction force is the gradient of the attraction field:

$$F_{att} = -\nabla U_{att}(q) = -K_a|q - q_d| \tag{3.10}$$

Unlike attraction force, it has generally been recognised that a repulsive potential should have limited range of influence. This prevents an object from affecting the motion of the robot when it is far away from the object. The repulsion field and the corresponding force are expressed as follows:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}K_r(\frac{1}{\rho} - \frac{1}{\rho_o})^2 & \rho \leq \rho_0 \\ 0, & \rho > \rho_0 \end{cases} \tag{3.11}$$

$$F_{rep} = -\nabla U_{rep} = \begin{cases} K_r(\frac{1}{\rho} - \frac{1}{\rho_o})(\frac{1}{\rho_0}) & \rho \leq \rho_0 \\ 0, & \rho > \rho_0 \end{cases} \tag{3.12}$$

Artificial Potential Function suffers from oscillation and local minima. Oscillation and local minima occur when the attractive force and the sum of repulsive forces are equal or almost equal and collinear but in the opposite direction moving to the goal location (Li et al., 2012). Zero-sum force causes the robot to oscillate (see Figure 3.12(a) and (c)). When local minima occurs near the goal, the robot cannot reach the target (see Figure 3.12(b))

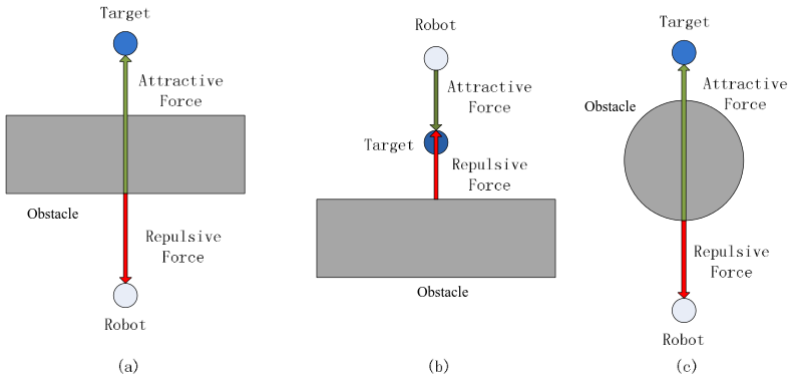


Figure 3.12: Local minima (Li et al., 2012).

Overcoming local minima and oscillation have been investigated extensively, and as a result, various methods have been suggested. Okamoto and Akella (2016) classified the proposed solutions into two categories: (i) solutions which aim to eliminate the local minima by using special potential functions including navigation function (Koditschek and Rimon, 1990) and harmonic function (Connolly and Grupen, 1993). And (ii) solutions which aim to resolve the local minima after the robot is trapped. There are a variety of approaches in this category including applying additional forces (Li et al., 2011), Regression search method (Li et al., 2012), placing a virtual obstacle (Park and Lee, 2003), virtual goal (Yang and Dong, 2012), and applying random movement direction (Lee et al., 2012). These approaches provide promising results, and they can be applied to the current research.

The cpp code for collision avoidance with the artificial potential function using laser

range finder in ROS can be accessed in Appendix B. The code is implemented in Stage simulation environment (Janani, 2017) [2] and on Pioneer-3AT robot (Janani, 2018a) [3] and (Janani, 2018b) [4].

3.6 Challenges in Obstacle Detection Implementation

When implementing the artificial potential function collision avoidance in the team, few team-level concerns appear which have to be addressed.

3.6.1 Differentiation between other team members and the rest of the obstacles in the environment

This valuable piece of information assists robots to decide more efficiently on the required action when encountering other robots. In other words, by recognising other robots in the field, robots can understand other robots intention and perform collision avoidance in a more efficient way.

There are various approaches with or without direct communication to assist robot to differentiate between other robots and obstacles in the environment. However, two communication free approaches are identified. These two approaches utilise

[2]APF Simulation Video Link:

[3]APF in Action Video Link:

[4]APF in Action Video Link:

the attached sensory system (i.e. laser range finder and 2D camera), hence do not require to add any more hardware to each individual.

Object Profiling Using Laser Range Finder Laser rangefinders return data in a dense format. For example, Hokuyo RG-04LX-UG01 returns 683 beams in 240^o field of view meaning that the angle between two consecutive beams is 0.3515625^o. (Hokuyo, 2009). When encountering an object multiple range points will represent the object. The objective in here is to classify these points into different objects and correlate the objects with pre-defined models and separate those that are close to the definition of other robots.

One of the most popular approaches is clustering. Clustering is one of the most widely used techniques for exploratory data analysis (Shalev-Shwartz and Ben-David, 2014). Shalev-Shwartz and Ben-David in 2014 describe in details various methods for clustering. In here, connectivity-based cluster finding is implemented since the number of points is limited.

In this approach, first, the acquired range data, which is returned in the polar system of coordinate, is converted into pointclouds, which are in the 2D Cartesian system of coordinate. Forming the first cluster at point zero $p[0]$, for every point, Euclidean distance to the next immediate neighbour is calculated. If the distance to the next neighbour is smaller than a pre-defined parameter called $maximum\_cluster\_distance$, the point will be included in the current clusters, otherwise, the previous cluster will be closed and a new cluster will be created. The closed cluster will be discarded if the size of the cluster, i.e. the number of points in the cluster, is less than another predefined parameter called $min\_cluster\_size$.

Cluster finding class and other laser analysis ros-package is available in github through `https://github.com/ajanani85/laser_analysis.git`. In addition, the

cpp code for cluster finding is in Appendix C and D.

Team-mate Recognition Using 2D Camera In teammate recognition using 2D vision sensor, robots search for specific 2D features on their acquired images. The feature could be a mounted artificial feature (e.g. QR Code, Fiducial Tag, etc) or predefined 2D template of the operating robots. They bring different benefits and drawbacks. One main common benefit of both approaches is that with the help of OpenCV solvePnP and Rodriguez function, you can get distance and orientation of the detected feature.

Using QR-Code, Fiducial Tag recognition (including Chilitags (Q. Bonnard and Dillenbourg, 2013), Apriltags (Wang and Olson, 2016), Aruco (Garrido-Jurado et al., 2014), and etc), or any other artificially mounted features one can identify and distinguish the robots from each other while increasing the design time. This in the future can benefit the team if, for example, the group aimed to be divided into smaller ones and masters are required. In this scenario, the robots can find their masters by searching for particular tag id.

On the other hand, template finding approaches, including SIFT, SURF, FAST, and BRIEF algorithms requires minimum design time, however, it does not provide differentiation between robots in a homogeneous team of robots and it is, in terms of computation time, expensive.

In this research, two approaches have been implemented and tested:

Chilitags and QR-Code: In separate approaches, the Chilitags and QR-Codes were mounted on the body of the robots as demonstrated in (3.13) and upon detection, the robot be able to find the position of the tag using solvePnP and Rodriguez (in OpenCV) in the attached camera's frame of reference. Since, QR-codes, specifically, can carry up to 4000 characters, certain messages could

be encoded with which certain behaviour could be triggered on other robots.

Figure 3.13: The displayed QR-code is causing the rear robot to follow the front one

The main problem with both the QR-code and fiducial tags is that both have limited range of detection and they require a direct line of sight. In particular, using 15 cm by 15 cm fiducial tag, the detection range with VGA resolution camera (480 X 640) is only 3.0 meters at detection frequency of 20 Hz. Series of tests conducted in different lighting conditions proving the claim.

On the other hand, as the distance increases, error in the evaluated orientation resulted from cv::solvePnP and cv::Rodriguez increases. This is due to the flickering of one or more pixels at the corners which will result in different 3D orientation estimation. As it is demonstrated in 3.14, a mistakenly included or excluded pixel in either direction (x or y) will result in a different 3D representation of a 2D image. This will be intensified if the observer is in motion. The impact of this phenomena is highlighted when it is used to determine the position of the observer. Generally, to the position of the observer (or in this case the camera) can be determined using the following:

$$\begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} = R^{-1} * \begin{bmatrix} X_t \\ Y_t \\ Z_t \end{bmatrix} \tag{3.13}$$

Where, o refers to the observer, t is the perceived tag position, and R is the perceived rotation matrix of the tag.

Series of tests have been conducted to estimate the variation in the estimated position of the observer and the tag. In this test, the camera is set at particular distances toward the tag and a software record the position of the tag and calculate the corresponding camera position using the equation 3.13. The results of the field test is demonstrated in Table 3.1.

As it can be seen, the phenomena a significant effect in X and Y direction on camera pose estimation and at 3.00 meters it varies more than 50 cm.

Table 3.1: Variation in tag position and observer position vs Distance to the tag

Distance (m)	Variation in Tag Position (m)		Variation in Camera Position (m)	
	ΔX	ΔY	ΔX	ΔY
0.50	0.0007	0.0007	0.001	0.001
1.00	0.0009	0.0008	0.010	0.0090
1.50	0.0012	0.0011	0.025	0.030
2.00	0.0020	0.0017	0.050	0.045
2.50	0.0030	0.0025	0.120	0.100
3.00	0.0057	0.0060	0.65	0.50

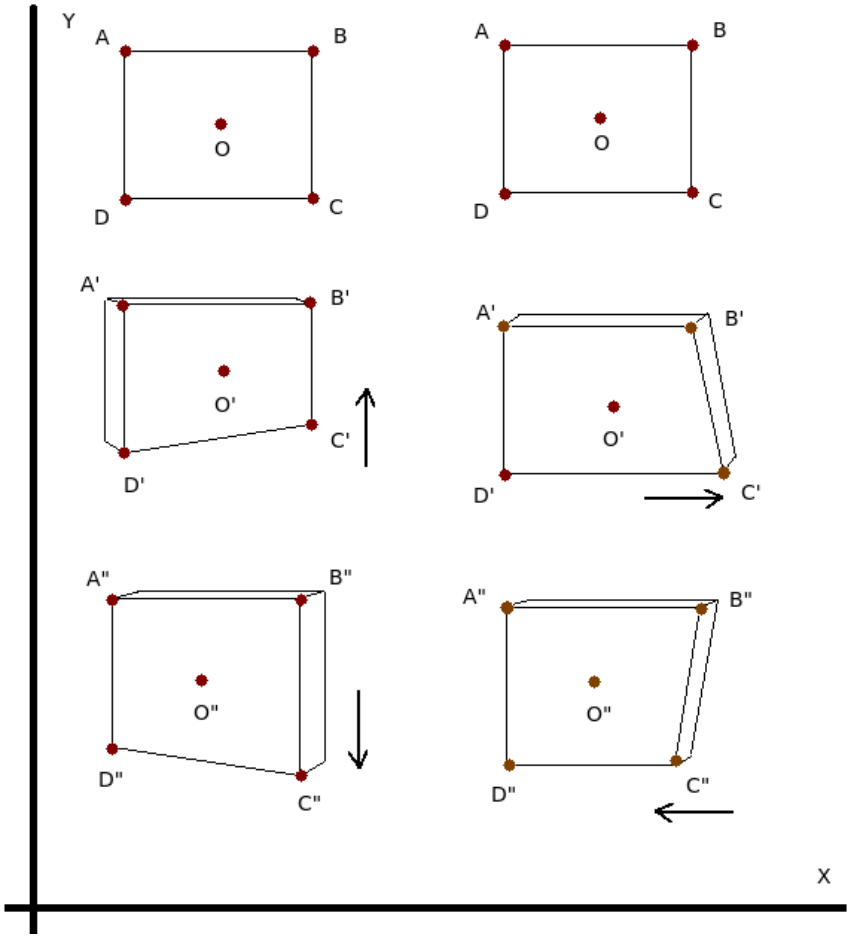


Figure 3.14: The flickering pixels at corners will cause different orientation estimation in 3D

With this consideration, it is recommended to rely only on the 3-DOF position of the tag. However, if one aims to utilise the orientation of the tag in the calculation, including estimating the orientation of the vehicle or position of the observer, either the camera has to be fixed with a gimbal so that the camera motion is compensated or a synchronized IMU has to be fused with the captured image.

RGB-LED pattern Recognition: Fiducial tags are passive features, meaning that the detection result is sensitive toward changes in environmental lighting. Using active source features will reduce this effect and improves the detec-

tion. To test this idea, RGB-LED-based patterns are created and mounted on robots, and appropriate software is designed to detect the pattern (refer to Appendix F).

This form of feature detection is implemented in indoor localisation project described in (Prez, 2016) supervised by the author and Dr. Lyuba Alboul at Sheffield Hallam University. In this project, a particular RGB-LED pattern demonstrated in Figure 3.15, is mounted on top of the Khepera III robot(K-Team, 2017).

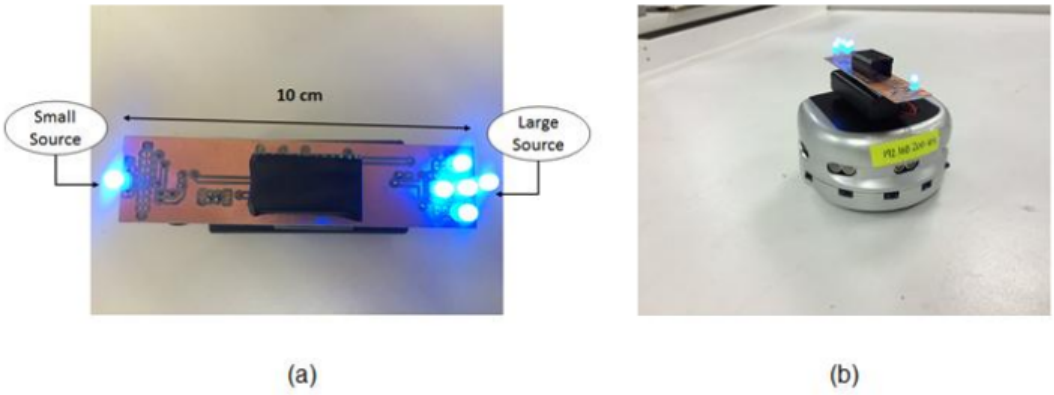
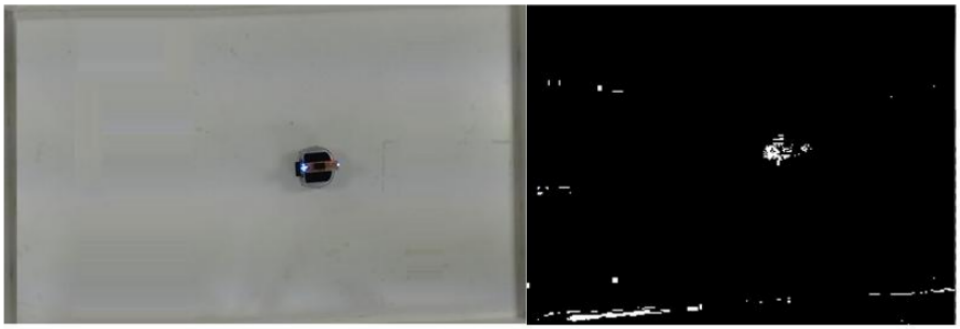


Figure 3.15: RGB-LED Feature Localisation: (a) The Pattern contains a big-cluster and small-cluster, (b) The feature configuration on Khepera III robot

The vision system consists of a 1080p Logitec camera c920 mounted on the ceiling and a detection software. The feature detection software and localisation, which is based on the HSV color detection, first converts the acquired RGB image to the HSV colorspace. Due to the noise in the environment, the resulting image contains many points with the similar filtered color characteristics (Figure 3.16.b).



(a) (b)

Figure 3.16: RGB-LED Feature Based Localisation Camera View: (a) 1080p Logitec camera c920 mounted on the ceiling, (b) HSV-based color detection result contains noise

To remove the small white spots, morphological filters are applied and the image is opened: that is to erode the image (Figure 3.17.a) first followed by dilation (Figure 3.17.b).

(a) (b)

Figure 3.17: RGB-LED Feature Based Localisation Noise Removal: (a) Erosion result, removing small white spots from the frame (b) Dilation result, emphasising the small spots left after erosion.

According to Prez (2016), the maximum errors in position is 6.9352 (mm), and 1.6183 degrees in the heading.

3.6.2 Entering the Field due to Collision Avoidance

As robots get closer and closer to the field, the resulting collision avoidance sum vector may force them to enter the field. This is unacceptable and has to be avoided because of two main reasons: 1. robots in the field are not performing collision avoidance, and 2. the already created patterns on the soil will be run over. As robots are performing the task of ploughing, they are not performing APF-based collision avoidance. This is due to the fact that any evasive manoeuvre inside the field will make the robots exiting their trajectory, destroying the created pattern, and running over the ploughed section. Therefore, this is non-ploughing robots' responsibility to take action.

To prevent robots from entering the field, an artificial repulsion force is applied at the borders of the field. Any robots that are closer than a particular threshold distance to the borders will consider this force. Therefore, the sum force near the field borders will look like the following:

$$F_{sum} = F_{att} - \sum_{i=1}^{n} F_{REP_i} - F_{BRep} \tag{3.14}$$

3.6.3 Combination of Collision and Congestion Avoidance

It is clear that robots have to avoid collision and resource conflict at the same time. However, there is one more problem: robots have to apply the resource conflict clearance behaviour only when they encounter another robot. In other words, in order for this approach to be successful, robots have to differentiate each other from other objects in the environment. But, this is still not sufficient since a faulty robot might create congestion in the field. Therefore, the status of the detected robot has to be determined too.

Surely, if robots had the capabilities to broadcast their status to the nearby robots, the problem becomes trivial. But robots are only capable to interact with each other via implicit forms of communication. One way to resolve this is to analyse the behaviour of the detected object by monitoring its activities using the attached range sensors. This approach is noise prone and may not always lead to success. Another method is to avoid collision until certain distance toward the shared location is reached, and switch to resource conflict clearance while within this threshold distance (see Figure 3.18). In this research, the latter approach is implemented.

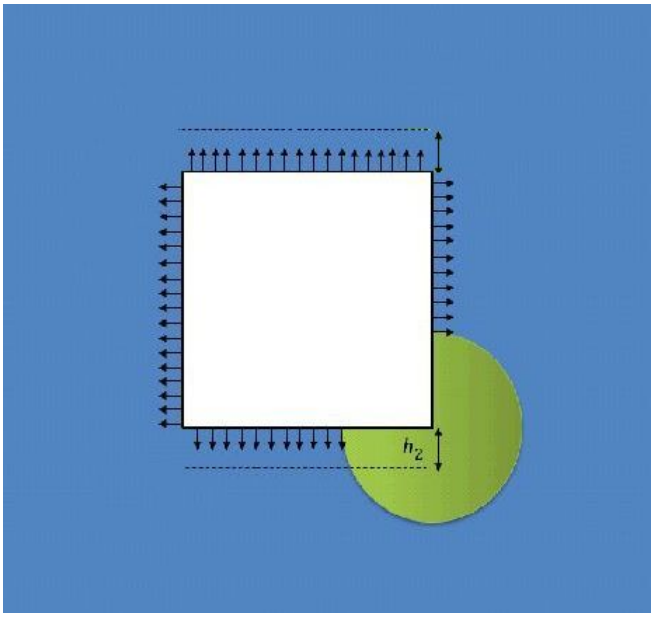


Figure 3.18: The field applies repulsion to prevent robots from entering while approaching the first ploughing location. Robots in the blue region avoid collisions only. Robots in the green region avoid resource conflicts only.

3.7 Team Ploughing

Regardless of the type of ploughing mouldboard, furrows are created the same way. As mentioned, ploughing is a two-dimensional navigational process. If a robot ploughs a field at a constant speed, the required time for a robot to complete ploughing a furrow is

$$t_p = \frac{l_f}{v} \tag{3.15}$$

Where l_f is the length of a furrow and v is the constant velocity of the robot.

When in a team, the robots have to plough in a queue as illustrated in Figure 3.19. This is because furrows have to be created consecutively. Assuming that robots are informed where to plough, they have equal length (λ), and there is an equal minimum distance between robots (ϵ), they plough the field with equal constant velocity, v. Thus applying a team introduces a delay time, as each robot except the first one has to wait for its predecessor. The delay time required for each robot can be calculated as follows:

$$t_{delay_i} = \frac{(i-1)(\lambda+\epsilon)}{v} \tag{3.16}$$

This delay is generated only once and propagates through to the following round of ploughing. Therefore, ploughing time for a robot is obtained as follows:

Corollary 1: The time to plough the ploughable area is

$$T_{ploughing} = K.t_p + t_{delay_n}$$

3.8 Furrow Transitioning

How to determine the first location to plough is one problem, but where to plough next is another. As emphasised in section 3.1.4, with reversible mouldboards it is possible to plough the field more efficiently as furrows can be created consecutively from both sides. Let's assume that robots are equipped with reversible mouldboards, and they are capable of ploughing from both directions. In order to utilise this capa-

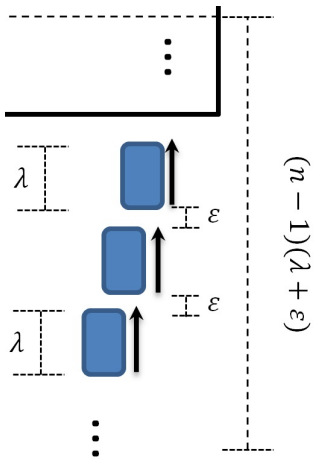


Figure 3.19: Team of n robots are aiming to plough. In the best case scenario, if the robots have formed the queue initially.

bility, upon finishing ploughing a location, robots have to standby in the headland area waiting for the last participating robot to complete its current furrow. In this situation, *where to plough next* depends directly on how robots behave in headland during furrow transitioning. In other words, furrow transitioning dictates how to distribute furrows among robots. According to this, two furrow transitioning models have been identified, and they are presented in the following sections.

3.8.1 Ploughing with a Reversible Mouldboard: First-In, First-Out

Let's assume that there is an initial order in a team of n robots. The order for the first shift is $robot_i$ is assigned with $furrow_i$. In FIFO, robots aim to maintain the original order throughout execution. As the number of robots is smaller than the number of furrows ($n < K$), robots may need to plough more than one furrow. If robots aim to maintain the original order in the team and if the task is distributed among robots always from r_1 to r_n, then the number of times that r_i has to perform ploughing can be obtained as follows:

$$k_i = \begin{cases} \lceil \frac{K}{n} \rceil & \text{if } i \leq (K \mod n) \\ \lfloor \frac{K}{n} \rfloor & \text{if } i > (K \mod n) \end{cases} \tag{3.17}$$

In here, $\lceil \frac{K}{n} \rceil$ represents the smallest following integer, and $\lfloor \frac{K}{n} \rfloor$ represents the largest previous integer.

Knowing k_i is necessary but it is not sufficient as the ploughing locations allocated to r_i are distributed throughout the field and they are not next one another. For that, if R is the set of robots, $R = \{r_i | i \in \{1, 2, ..., n\}\}$, and F is the set of furrow numbers in the field, $F = \{f_l | f_l \in \{1, 2, ..., K\}\}$, and G_i is the set of ploughing locations allocated to r_i, where

$|G_i| = k_i$, and

$\bigcup_{\forall i | i \in \{1,2,...,n\}} G_i = F$, and

$G_\alpha \bigcap_{\alpha,\beta \in \{1,2,...,n\}, \alpha \neq \beta} G_\beta = \phi$,

then ploughing locations (or furrow number) allocated to r_i is:

$$G_i = \{f_{i+jn} | j \in \{0, 1, ..., (k_i - 1)\}\} \tag{3.18}$$

During FIFO headland navigation, robots have to achieve three goals: (1) maintaining the original order, (2) reaching the next ploughing location, and (3) adjusting the heading by 180°. As f_j always have to be processed before f_{j+1}, the robot that is responsible for processing f_{j+1} is not allowed to enter the field before the robot which is allocated with f_j. This means that r_1 cannot start the ploughing before r_n exited the field.

To maintain the original order, in FIFO, robots stack behind r_1 until r_n exits the field. To achieve this, the robots have to form a loop, as illustrated in Figure 3.20. Upon completing a furrow, each robot joins the queue in the direction of the (virtual)

point A in Figure 3.20. Then the robot turns to B and C respectively. Where C is in the line of the next furrow. Ploughing starts at point D, as soon as there is space (the last robot has left, or r_{j-1} has begun).

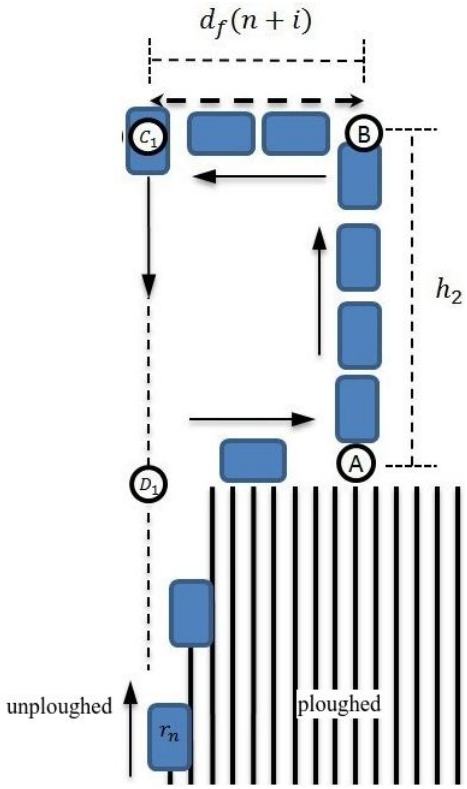


Figure 3.20: First-In, First-Out Furrow Transitioning: (A) Move to the last position of r_1. (B) Approach orthogonal projection of point A on headland. (C) Move to your next starting point orthogonal projection on the margin of the field. (D) Next Starting point for r_1.

Time analysis of furrow transitioning is carried out in two stages: The time for a robot to travel the designated distance (t_{fft_i}), and the waiting period during furrow transitioning (t_{w_i}).

In the first case, where there is no traffic to affect robots' locomotion, duration of furrow transitioning for a robot can be obtained from the sum of all 2-dimensional locomotion at each stage. For that, let's break down the distance that a robot travels during each stage. In the first stage, a robot travels to the position of the first robot.

This position is $d_f.(i-1)$ meters away from the current position of the r_i where i is the rank of the robot obtained during task allocation, and d_f is the distance between two consecutive furrows. r_1 can skip this stage since it is already at point (A). In stage 2, robots have to travel for the distance of h_2 meters, until they reach the lowest or the highest margin of the field. In stage 3, robots have to travel to their next allocated furrow in G_i. This position is $d_f.(n+i-1)$ meters away from point B for r_i. Finally, in stage 4, robots travel for another h_2 meters to reach the beginning of the ploughing location. Figure 3.20 demonstrates the described furrow transitioning pattern.

Considering that furrow transitioning has to be repeated on both sides of the field, it could be assumed that the width of the headland on both sides are equal ($h_2 = h_1 = H$). With this, the overall travelled distance that a robot needs to take for furrow transitioning without considering the impact of traffic is:

$d_{ft_i} = d_{stage_1} + d_{stage_2} + d_{stage_3} + d_{stage_4}$

$$d_{ft_i} = 2H + d_f.(n + 2i - 2) \tag{3.19}$$

Where, H refers to the minimum required headland, and it can be obtained as follows:

$$H = (\lambda + \epsilon)(n - \lfloor \frac{d_f.(2n-1)}{\lambda + \epsilon} \rfloor) \tag{3.20}$$

Derivation: Let's assume that the length of all robots are equal, and it is denoted by λ. Let's also assume that the threshold distance between two robots is denoted by ϵ. If a team of 10 robots is transiting their furrows, between the position of last robot (r_{10}) and point (A) there are 9 furrows. Also, between point (B) and point

(C), which is the next furrow position for r_1, there are 10 furrows. Therefore, the sum of distances in both stages is $19.d_f$ meters. With this, it is possible to identify the number of robots that can fit in these two sides: $\lfloor \frac{19.d_f}{\lambda+\epsilon} \rfloor$. The length that the rest of the robots, $n - \lfloor \frac{19.d_f}{\lambda+\epsilon} \rfloor$, require to fit between point (A) and (B) is the minimum width for the headland, $(\lambda + \epsilon)\lfloor n - \frac{d_f.(2n-1)}{\lambda+\epsilon} \rfloor$.

When d_{ft_i} is known, it is possible to evaluate the time required for a robot to perform furrow transitioning in a traffic-free manner.

$$t_{fft_i} = \frac{d_{ft_i}}{v} \tag{3.21}$$

As mentioned, robots have to wait in the headland area until the last robot exited the ploughing area. This means r_n has a direct impact on the waiting period of each robot. In order to determine the effect of r_n's ploughing time on r_1's furrow transitioning time, one needs to take into account that during the time that r_n is ploughing, r_1 is also performing ploughing and furrow transitioning, and by the time r_1 reaches its waiting point (which is point C in Stage 3), r_n has already completed some portion of its task. If the task is performed without any collision or congestion and at a constant speed, the effect of the ploughing time of r_n on furrow transitioning time of r_1 can be evaluated as follows:

$$t_{w_1} = t_{p_n} - t_{p_1} - \frac{H + n.d_f}{v} \tag{3.22}$$

In here, $\frac{H+n.d_f}{v}$ refers to stage 2 and stage 3 of furrow transitioning performed by r_1, v is the velocity of the robot, t_{p_n} and t_{p_1} are the ploughing period for robot r_n and r_1 respectively.

Other robots, $r_{i>1}$, will travel shorter distances as they stack behind r_1. With this

consideration standby time, t_{w_i}, for robots can be obtained as follows:

$$t_{w_i} = t_{p_n} - t_{p_i} - \frac{H + n.d_f - (i-1)(\lambda + \epsilon)}{v} \tag{3.23}$$

From both, the furrow transitioning and traffic effect time, it is possible to determine the overall time required for a robot to perform furrow transitioning in a team with n robots in one round.

$$t_{ft_i} = t_{w_i} + t_{fft_i} \tag{3.24}$$

If r_i is required to plough k_i furrow, it has to perform furrow transitioning for $k_i - 1$ times. Therefore, r_i's overall furrow transitioning duration is $(k_i - 1).t_{ft_i}$.

Figure 3.21 demonstrates the described algorithm in the form of flow chart.

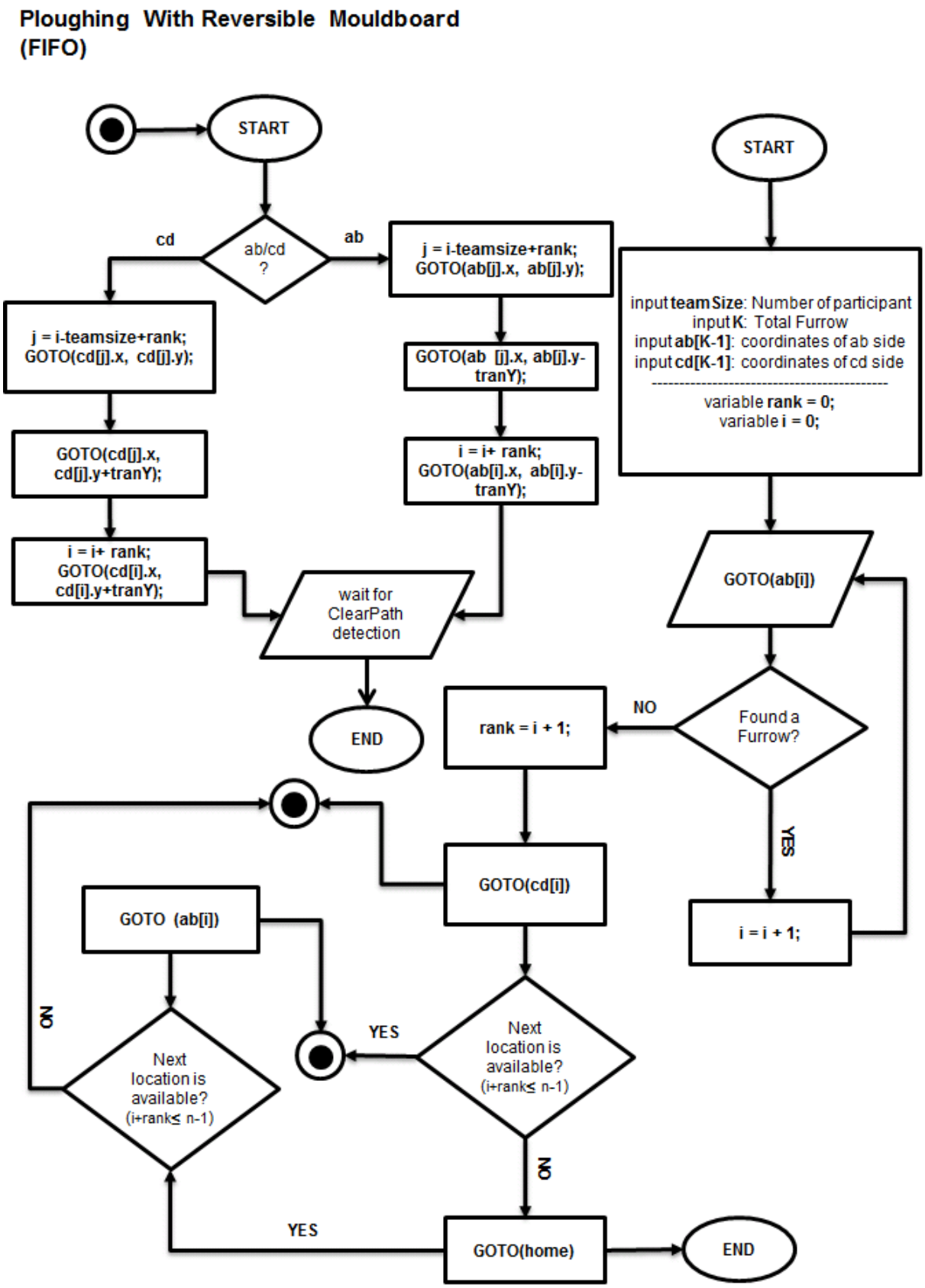


Figure 3.21: Ploughing with a Reversible Mouldboard FIFO - Flow Chart

3.8.2 Ploughing with a Reversible Mouldboard: Last-In, First-Out

It can be seen in equation 3.20 that the width of the headland depends heavily on the number of robots in the team: the size of the ploughable area will shrink by the increase in the number of participating robots. This is a major drawback of FIFO since it affects the profitability of the field (there are many other applications which can still utilise this model with the current results). Therefore, we sought for an alternative approach by which the more headlands could be limited.

In LIFO, the main aim is to preserve more headlands. One way to improve FIFO approach is to introduce a more flexible task allocation method. Similar to FIFO, let's assume that the order of the first round of ploughing among n robots is $robot_i$ is assigned with $furrow_i$. Instead of maintaining this order for next round, robots aim to reverse this order by letting the last robot to start first. This means that robots will change their order in the team to their $n - i + 1$ ranks. For example, if the original rank of a robot is 2 ($i = 2$) in a team of 5 robots ($n = 5$), in the next round of ploughing its rank will change to 4. As robots will be assigned with different ranks in the team, the number of furrows that each robot has to process (k_i) and members of robots' furrow sets (G_i) have to be redefined to compensate this change. Like FIFO, task is distributed from the r_1 to r_n. Therefore k_i is determined as follows:

if $\lfloor \frac{K}{n} \rfloor \in \{2, 4, 6, ...\}$:

$$k_i = \begin{cases} \lceil \frac{K}{n} \rceil & \text{if } i \leq (K \mod n) \\ \lfloor \frac{K}{n} \rfloor & \text{if } i > (K \mod n) \end{cases} \tag{3.25}$$

and if $\lfloor \frac{K}{n} \rfloor \in \{1, 3, 5, ...\}$:

$$k_i = \begin{cases} \lfloor \frac{K}{n} \rfloor & \text{if } i \leq (K \mod n) \\ \lceil \frac{K}{n} \rceil & \text{if } i > n - (K \mod n) \end{cases} \tag{3.26}$$

Also, members of G_i can be identified by the following:

$\forall j \in \{1, 2, ..., k_i\}$

$$f_j = \begin{cases} i & \text{if } j = 1 \\ f_{j-1} + 2(n-i) + 1 & \text{if } j = 2p, p \in \mathbb{Z} \\ f_{j-1} + 2i - 1 & \text{if } j = 2p+1, p \in \mathbb{Z} \end{cases} \tag{3.27}$$

With order of a robot changing at every round, the ploughing time of a robot also is affected:

$\forall j \in \{1, 2, ..., k_i\}$

$$t_{p_i} = \begin{cases} \frac{l_p - 2H + (i-1)(\lambda+\epsilon)}{v} & \text{if } j = 2p+1, p \in \mathbb{Z} \\ \\ \frac{l_p - 2H + (n-i)(\lambda+\epsilon)}{v} & \text{if } j = 2p, p \in \mathbb{Z} \end{cases} \tag{3.28}$$

The furrow transitioning that satisfy this condition requires that r_i instead of stacking behind of r_{i-1}, stack in front of it. With this, the robot that was originally assigned with rank i will be re-ordered in the team by $n - i + 1$. For example, r_1 becomes r_n in every other round.

To implement this, a three-stage procedure is designed, and robots as long as their order during the ploughing are not the last robot in the team ($i \neq n$) has to comply

with these stages. In the first stage, robots have to keep rotating to the left (during furrow transitioning at h_1) until their heading difference becomes 90°. If the length of a robot and its extension is denoted by λ, then a location with $(x_c - \lambda, y_c + \lambda)$ coordinate could be assigned as a target for the robot (where (x_c, y_c) is referring to the furrow ending coordinate). This location is marked by (A) in Figure 3.22.

In stage 2, the robot maintains its heading and moves for double of its length reversely. This point can be addressed by $(x_c + 2\lambda, y_c + \lambda)$ coordinate in the field (refer to (B) in Figure 3.22. During its approach to point B, a robot does not necessarily have to observe the behind area. Instead, a robot that has already reached the point (B), scans for any critically closing objects. Upon detection of a reversely approaching robot, the detector robot will move reversely to maintain the critical distance. The last robot that joins the queue is r_{n-1}. In addition to moving reversely, a robot that reached point B has to monitor the behaviour of the front robot for initiation of stage 3. This could be carried out by a range based sensor or a simple camera.

Stage 3 starts when r_{n-1} detects that r_n which becomes the first robot for a new round clears the path. r_n, instead of complying with the stages, can directly access its next ploughing starting coordinate as there are no other robots in the field. Once r_n moves out of the field of view of r_{n-1}, it starts approaching its next round of ploughing. Other robots behave the same as soon as they detect such behaviour from the front robot.

During Stage 1, robots travel equal distances if they have equal dimensions. The travelled distance at this stage can be evaluated as follows (let λ be the length of the robot and):

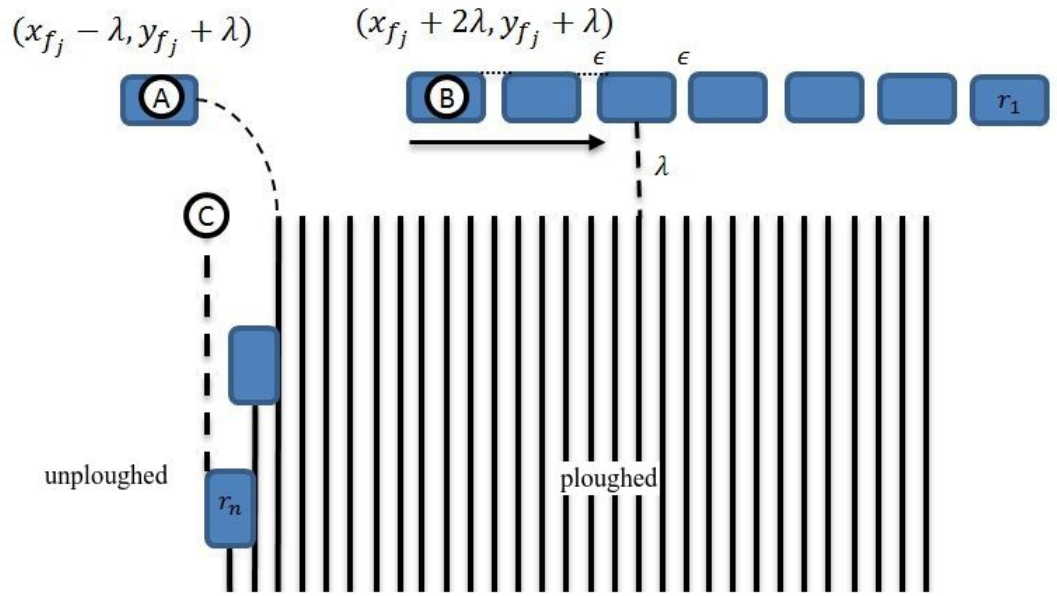


Figure 3.22: Three stage procedure of ploughing with a reversible mouldboard LIFO

$$d_{stage1_i} = \lambda.\sqrt{2} \tag{3.29}$$

In Stage 2, each robot will travel two different sets of distances: (1) distance to reach the designated point B, and (2) distance to avoid collision with the front robot(s). In the first part of stage 2, each robot has to travel 2λ meters to reach the location of point B. However, in order to obtain the travelled distance in part 2, a deeper analysis is required. The initial distance difference between two consecutive robots is equal to the distance between two adjacent furrows (d_f). And as point A and B's coordinates for robots are relative to their end ploughing locations, then point A and B of two consecutive robots are d_f meters away from each other. If distance between two consecutive furrows, d_f, is smaller than the length of the robots, λ, and the threshold distance between two robots is denoted by ϵ, then the total distance that a robot with complementary rank i has to travel is:

$$d_{stage2_i} = (n - 1 - i)(\epsilon + d_f) + 2\lambda \tag{3.30}$$

In stage 3, once the path is cleared, robots have to travel to their next starting point. The travelling distance consists of the travelled distance during stage 2 to reach to the initial point, and the distance to reach the new ploughing location:

$$d_{stage3_i} = d_{stage2_i} - \lambda + d_f.(n + i + 1) \tag{3.31}$$

With travelled distances at each stage known, it is possible to identify the duration for r_i to perform furrow transitioning without consideration of traffic effect:

$$t_{fft_i} = \frac{d_{stage1_i} + d_{stage2_i} + d_{stage3_i}}{v} \tag{3.32}$$

Like FIFO, robots execution affects one another. However, this effect is compensated by the time that each robot spends to reach stage 2 of furrow transitioning. Because of the re-ordering, the effect of the traffic will also change for each robot. This effect for each robot can be evaluated as follows:

$\forall j \in \{1, 2, ..., k_i\}$

$$t_{w_i} = \begin{cases} t_{p_n} - t_{p_i} - t_{stage1_i} - t_{stage2_i} & \text{if } j = 2p + 1 \\ \\ t_{p_1} - t_{p_{n-i+1}} - t_{stage1_{n-i+1}} - t_{stage2_{n-i+1}} & \text{if } j = 2p \end{cases} \tag{3.33}$$

With this approach, robots require less heading width for obtaining the required orientation:

$$H = \zeta + \lambda \tag{3.34}$$

Where ζ is the width and λ is the length of the robot. Figure 3.23 demonstrates the described algorithm in the form of flowchart.

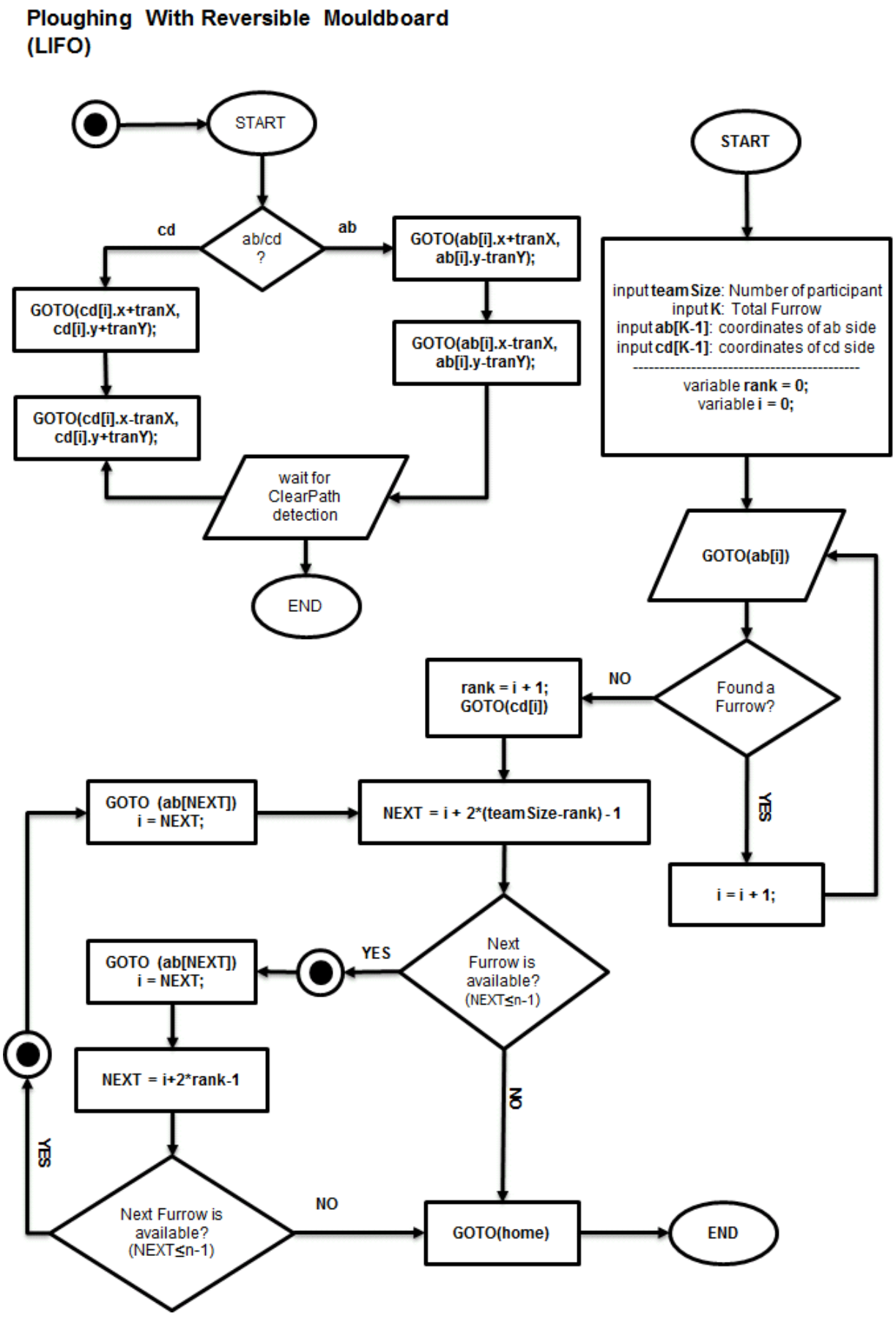


Figure 3.23: Ploughing with a Reversible Mouldboard LIFO - Flow Chart

3.8.3 Comparison and Discussion

In the previous section, two furrow transitioning method (FIFO and LIFO) along with the corresponding task partitioning and allocation were discussed. The productivity and efficiency of the proposed approaches can be indicated via three parameters: (i) the length of the ploughed furrow (l_p), (ii) team ploughing execution time, and (iii) the required travelled distance. (i) indicates the amount of crop that can be grown in the field (yield), whereas (ii) and (iii) indicate the ploughing cost. For mathematical visualisation and simulation purposes, a series of environmental parameters have to be set. The details of the field are demonstrated in Table 3.2.

Table 3.2: List of parameters for simulation and mathematical visualisation

Parameter Name	**Values**
Field Dimension ($W \times L$)	$20(m) \times 26(m)$
Distance Between Two Consecutive Furrows (d_f)	$40(cm)$
Length of a Furrow (l_p)	$20(m)$
Robot's Dimension ($\zeta \times \lambda$)	$0.5(m) \times 0.5(m)$
Threshold Distance between Two Robots	$0.25(m)$
Furrow Analysis Period	$5(s)$
Robot Velocity	$0.5(m/s)$

Furrow transitioning is performed in the headland area at both ends of the field. The width of the headland affects the length of the ploughed furrow:

$$l_f = L - 2H \tag{3.35}$$

Where, l_f is the length of the ploughed furrow, L is the length of the field, and H

is the width of the headland (assuming $h_1 = h_2 = H$). This means that the wider the headland, the shorter the length of the furrow, hence smaller cultivation area. Therefore, the productivity of a furrow transitioning method in terms of the length of the furrow can be expressed as follows:

$$\%Prod = \frac{l_p}{L} \times 100 \tag{3.36}$$

Using equations 3.20 and 3.34, the required headland, and the corresponding productivity can be predicted in different team sizes. A script is developed to calculate the productivity in both methods, and the results are demonstrated in Figure 3.24.

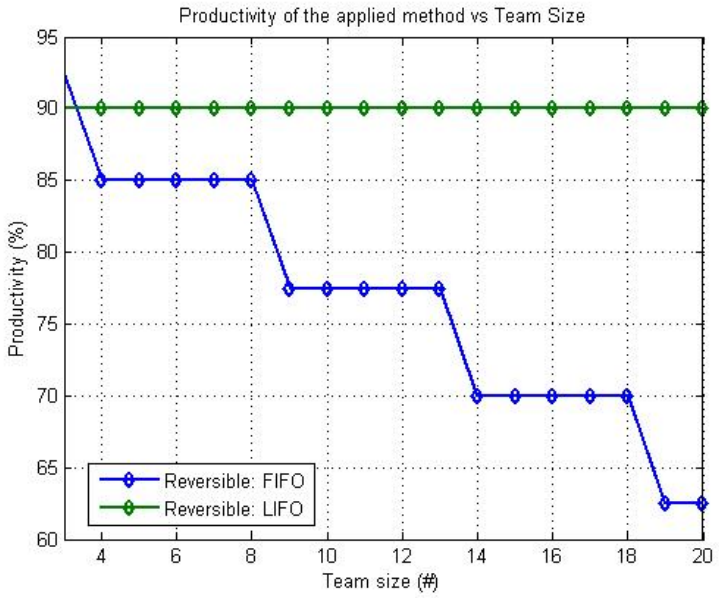


Figure 3.24: FIFO vs LIFO impact on productivity of the ploughing. By increase of number of participating robots in the team, the productivity in FIFO furrow transitioning method drops significantly. This is because the headland depends on the team size as described in equation 3.20. Whereas in LIFO, robots require consistent width for headland navigation, hence productivity remains uninfluenced by the team size.

According to Figure 3.24, the productivity in FIFO drops significantly by increase

in number of participating robots. Whereas, the productivity remains consistent in LIFO regardless of the team size (Only in a team of three robots FIFO becomes more productive).

Furthermore, the total ploughing execution time in the team and the maximum travelled distance that an individual will take during execution can be estimated and compared in different team sizes. Total ploughing execution time refers to the duration between the time that the first robot enters the field and the time that the last robot leaves the field. With this, series of simulations have been conducted to estimate the total execution time for both furrow transitioning methods in different team sizes.

As it is demonstrated in Figure 3.25 and Figure 3.26, in LIFO, robots perform furrow transitioning faster while travelling shorter distances. This reduces the cost of ploughing compare to FIFO furrow transitioning method.

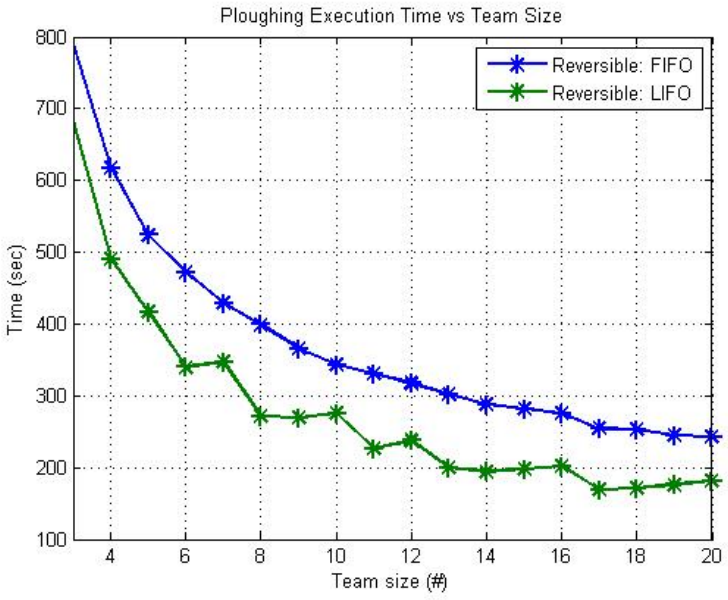


Figure 3.25: Ploughing time visualisation: team size $\in [3, 20]$. With LIFO, the team can plough the field much faster. The results are obtained from simulation conducted in Matlab.

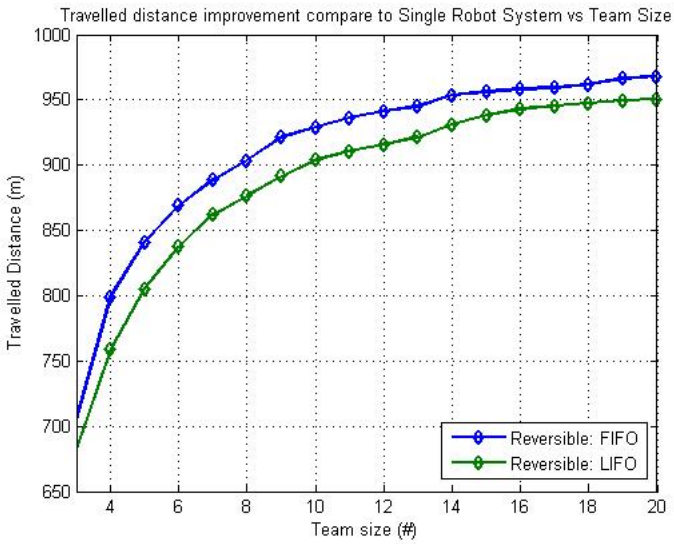


Figure 3.26: Ploughing distance visualisation: team size $\in [3, 20]$. In FIFO, robots have to travel longer distances for furrow transitioning. This increases the cost of ploughing.

Simulation In addition to mathematical visualisation, series of simulations were conducted in Stage simulation environment (Gerkey et al., 2003) in conjunction with ROS (Robot Operating System) (Quigley et al., 2009). In the robot simulation model, each robot is equipped with a forward facing Hokuyo laser range finder with 180.0 degrees field of view, a 32 x 32 pixel 2D camera facing forward pitched 45.0 degrees downwards, a localisation system providing the 3DOF (three degree of freedom) position of the unit in world model (x, y, yaw in radian), and a receiver unit for receiving task initiation signal. The simulation begins by broadcasting the start signal throughout the system.

The behaviour of each robot is being controlled by three C++ programs: **task-handler.cpp** which is responsible for decision making and selecting the target locations for the robot navigation, **reach-point.cpp** which utilises the information provided by the localisation system and the range sensor information to guide the robot to the requested location by avoiding collisions and resolving resource con-

flict. And **image-analyser.cpp** which is responsible for furrow detection. The task-handler.cpp communicates with other modules through ROS interprocess communication API. Figure 3.27 demonstrates the overview of the simulation control system.

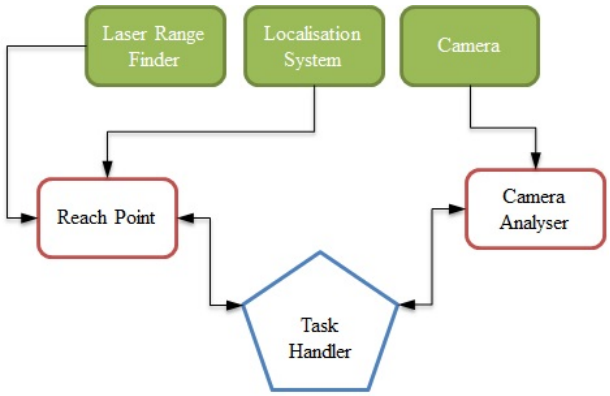


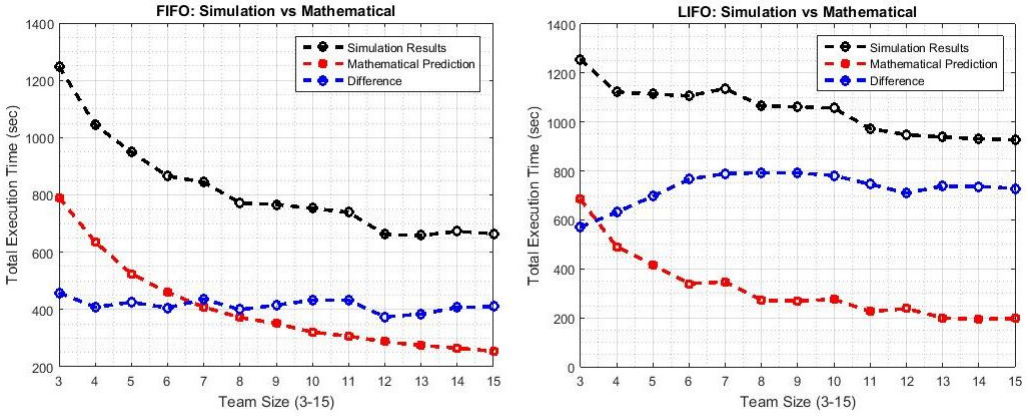
Figure 3.27: Overview of Stage simulation control system for each robot.

The videos for the simulations are in (Janani, 2015b) [5] for FIFO and (Janani, 2015a) for LIFO [6]. The simulations are run for different team sizes ([3 15]). Robots start the task simultaneously as soon as the programs start. During simulation, the timestamp along with the position of the robots are recorded separately while the robots are within 40×40 meters area. This is to observe the effect of simultaneous field accessing while robots are outside of the field. As demonstrated in Figure 3.28, the simulation results indicate similarities with results obtained from the mathematical description. However, there are differences between simulation results and mathematical visualisation which is due to the fact that mathematical descriptions do

[5]FIFO Simulation Video Link:

[6]LIFO Simulation Video Link:

not consider: the initial position of the robots, the existence of random velocity for congestion avoidance, and limited area of consideration ($1600m^2$ for simulations and $520m^2$ for mathematical visualisation).



(a) (b)

Figure 3.28: FIFO and LIFO Difference between Simulation and Mathematical Results: (a) FIFO Simulation (black colour) and Mathematical(red colour) Results. (b) LIFO Simulation (black colour) and Mathematical(red colour) Results. The blue line indicates the difference between simulation results and the results obtained from mathematical description.

All in all, if ploughing with a reversible mouldboard is desired, LIFO approach will provide faster and cheaper furrow transitioning while it maintains a fixed headland for different team size.

3.9 Ploughing Optimisation

The results obtained from FIFO and LIFO confirm the feasibility of cooperative ploughing. However, the proposed approaches have a few drawbacks. In this section, first, these drawbacks are discussed. Next, a new method is described by which cooperative ploughing becomes more resilient towards the dynamic changes that could occur during ploughing.

3.9.1 Issues with FIFO and LIFO

In FIFO and LIFO, in order for robots to perform furrow transitioning, robots have to consider the number of participating robots. In other words, the team size is fixed during execution time, and it can only be updated during the design period. This makes both approaches vulnerable to dynamic changes in the environment. Specifically, the loss of an individual during execution time results in incomplete ploughing. In other words, the system is not robust and it is susceptible to the loss of an individual. The worst case scenario occurs when the last robot fails. In both FIFO and LIFO, the last robot dictates when a new round of ploughing should be restarted as the rest of the team is standing by at the headland. In this situation, if the last robot fails to reach the headland, the rest of the team maintain their status. As a result, a significant portion of the field remains unploughed.

Moreover, the number of participating robots cannot be increased during ploughing either (i.e. the system is not scalable in real-time). For instance, if one robot is added to a team of 10 robots, without updating the other robots, ploughed locations will be redundantly ploughed. Therefore, increasing the team size during execution time has no impact on the processing time.

These limitations are due to the lack of communication among robots. Since the information regarding the team size cannot be updated during execution time, any changes in the team size (including increase or decrease) will impair the results. Robots should be able to find their next ploughing location independent of the team size.

In addition, robots are assumed to be equipped with only reversible mouldboards. This means that the system cannot tolerate any heterogeneity in the team. However, reversible mouldboards may not always be available. The system should be flexible

enough to adapt to these insignificant changes and tolerate heterogeneity in the team.

With these new design considerations, a new approach has to be investigated. An approach which is more flexible toward changes that could occur during execution. Instead of ploughing in a highly coordinated method, the cooperation should emerge as a result of individual executions. This is referred to as *emergent cooperation*. In emergent cooperation, robots do not explicitly work together (whereas in FIFO and LIFO, robots furrow transitioning depends on each other), but cooperation emerges as a result of their interactions with each other and the world and it is independent of team size (Gerkey et al., 2003).

3.9.2 Toward Self-Organising Ploughing

As mentioned, the most important drawback of FIFO and LIFO is task allocation's dependency on the number of participating robots which make the system susceptible to the loss of a single robot. Robots utilise the information about the team size to determine where to plough next. If robots plough the field from only one side (instead of both sides in FIFO and LIFO), the answer to the question of *where to plough next?* can be responded independently of the number of participating robots. In this approach, robots plough the field only from one side. After completion of a furrow, robots navigate back to the first ploughing location using the width of the field (see Figure 3.29).

As robots analyse the ploughing locations, their understanding of the task increases. For example, if a robot ploughs the third ploughing locations (f_3), for the next round it does not need to re-analyse the field up to the third ploughing location. Instead, the robots directly approach the fourth ploughing location (f_4).

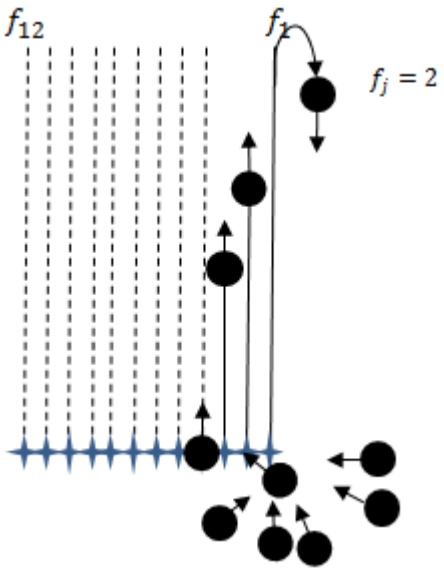


Figure 3.29: The self-organised approach: robots plough the field from one side only.

These learnings not only prevents redundant field analysis, but also it assists the robot to select the shortest path to the next ploughing location. If robot determines that the next ploughing location to analyse belongs to the second half of the field, it selects the opposite side of the field to reach to the targeted ploughing location (see Figure 3.30).

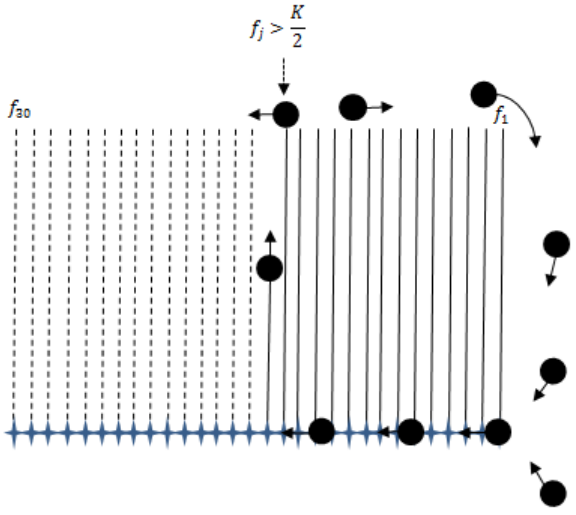


Figure 3.30: By building up the understanding of the environment, robots can select the shortest path to the next ploughing location.

A robot concludes that a field is ploughed when it finds the last ploughing location

processed. As soon as this conclusion is made, the robot clears the path for others to process to prevent congestion. Figure 3.31 demonstrates the flow chart of the self-organised ploughing, and Appendix I contains C++ task handler for the self-organised ploughing. (Janani, 2016) [7] contains the link to the video of simulation for self-organised ploughing.

In this approach, the task allocation is carried out in real-time and independent of the number of participating robots. Thus, the team becomes more robust toward the failure of one or more individual.

Also, the approach is more scalable since by increasing the number of participating robots, the individual behaviour will not be affected. In addition, this approach is independent of the attached ploughing mouldboard. Robots could be equipped with reversible or conventional mouldboard and yet they can contribute to the task.

However, one major drawback of this approach is that robots have to travel long distances for furrow transitioning to reach the next ploughing location. As a result, the cost of ploughing increases significantly. Using the same control architecture as FIFO and LIFO (Figure 3.27), series of simulations were conducted. As it is demonstrated in Figure 3.32, using the optimised ploughing approach robots have to travel farther distances. This increases the cost of ploughing significantly. However, by increase in the number of participating robots, this difference is reduced considerably. It could be concluded that the optimised ploughing approach has better efficiency when applied to a large team.

[7] Self-Organised Simulation Video Link:

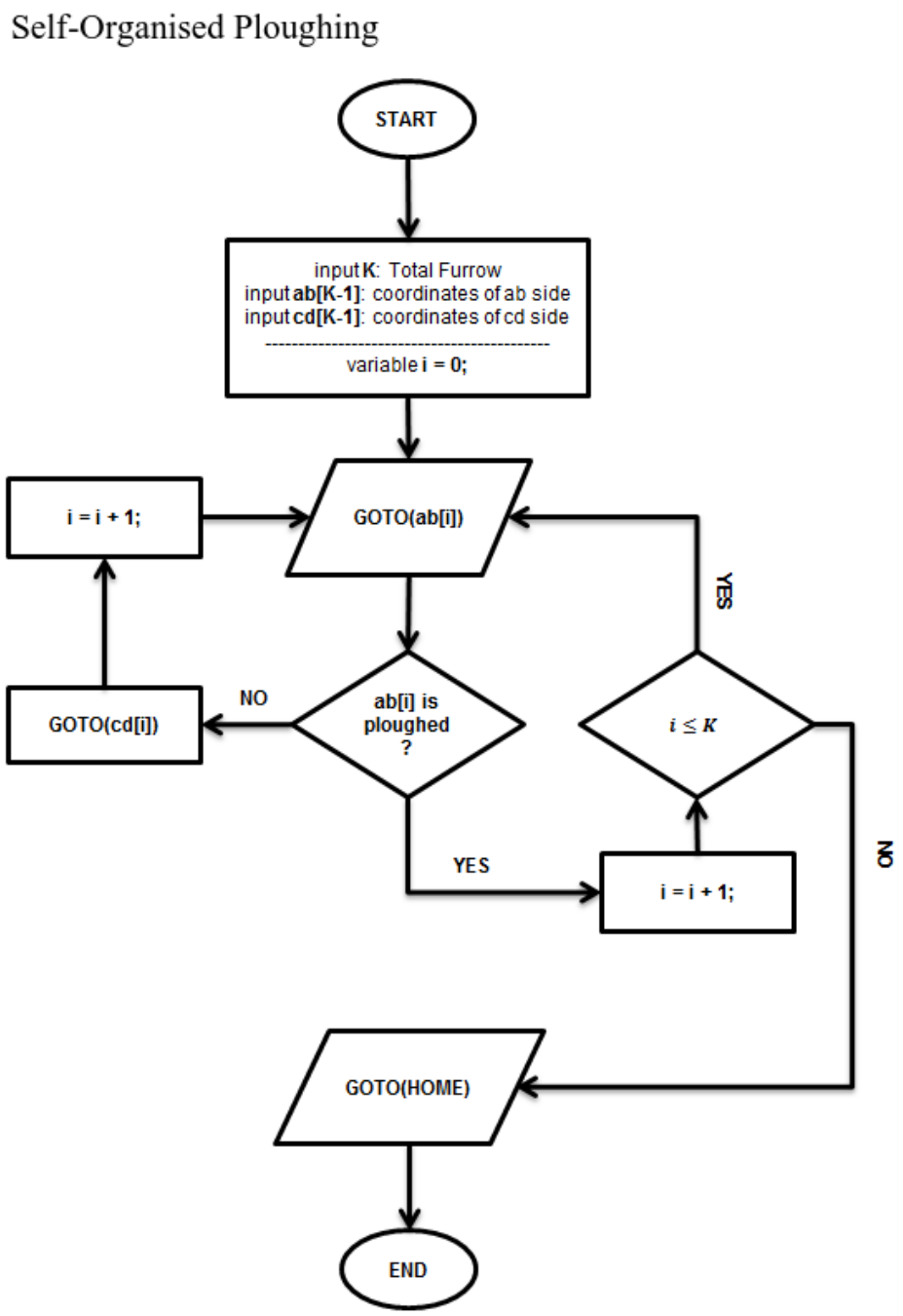


Figure 3.31: Self-organised ploughing flowchart

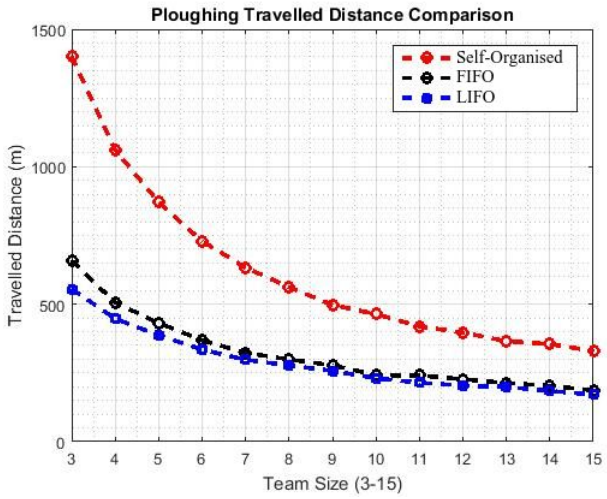


Figure 3.32: With the self organised method, robots have to travel longer distances. This increases the cost of ploughing.

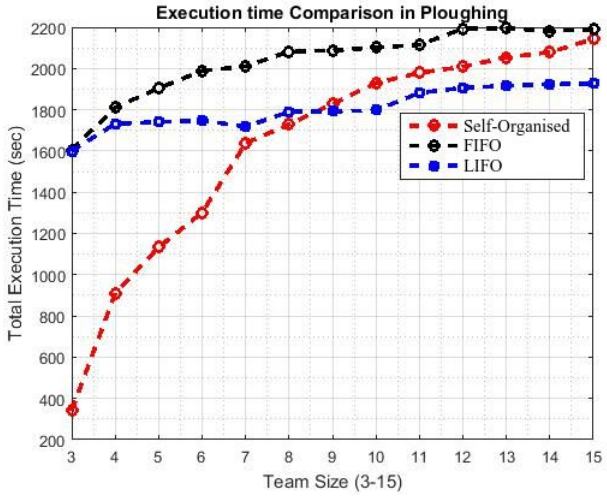


Figure 3.33: Time Analysis of Ploughing Methods: The impact of improvement on execution time (a single robot can plough the same field in $t_{single} = 2854.4(sec)$). By increase of number of robots in the team, the self organised ploughing method has better response.

3.10 Conclusion

In this chapter, the feasibility of cooperative ploughing in a team of robots is investigated. As a result, three cooperative ploughing approaches (FIFO, LIFO, Self-organised) are described, analysed and compared. In FIFO and LIFO, the aim is to utilise the capability of reversible mouldboard and perform ploughing from both sides of the field. The task allocation depends on the number of participating robots in the team, and this makes both approaches susceptible to the loss of a single robot. Since both approaches are not real-time scalable and robust system, any changes in the team size (including removal or increase of an individual) affect the end result. It could be said that FIFO and LIFO are effective only when the success of individuals can be guaranteed.

Therefore, the self-organising approach is proposed in which task allocation is carried out independent from the number of participating robots. The self-organising approach is not affected by changes in the team size (both increasing and decreasing the team size will not affect the system).

Even though the self-organising approach is more reliable, it increases the cost of ploughing since the robots have to travel relatively farther distances to reach the next ploughing location. The obtained results from simulation suggest that this effect becomes insignificant as the number of participating robots increases.

CHAPTER 4

COOPERATIVE SPRAYING: DESIGN AND IMPLEMENTATION

In this chapter, the feasibility of spraying a large field with a distributed team of robots is investigated. The motivation behind this chapter is presented in 4.1. The task of spraying is described and analysed in 4.2. The proposed approach for spraying in a team of robots are described in words, flow charts, and the implemented C++ codes in 4.3. The proposed approach is analysed, criticised , and an optimised approach is described in 4.5.

The points described in this chapter is published in the following:
Janani, A., Alboul, L. and Penders, J., 2016, June. Multi robot cooperative area coverage, case study: spraying. In Conference Towards Autonomous Robotic Systems (pp. 165-176). Springer International Publishing.

4.1 Motivation Behind Further Investigation

In the previous chapter, a scalable distributed cooperative algorithm was suggested for the task of ploughing. At first glance, it seems that the proposed solution can be applied to all agricultural tasks despite being classified as independent or sequential. One important point which is taken into consideration in cooperative ploughing is the physical changes that ploughing creates on the soil (i.e. ridges and furrows). Such patterns do not come into existence with other agricultural tasks.

Let's consider the task of spraying which starts after the task of ploughing. In this task, the furrows (which are now crop rows) are already created. If the same interaction method is applied(i.e. robots rely on detection of furrows to indicate if a location is processed), the spraying will never be initiated as robots perceive that the

task is completed, bear in mind that the existence of furrows are the indication for this. In other words, for spraying, the detected furrows carry insufficient information for concluding the state of completion of the task (see Figure 4.1).

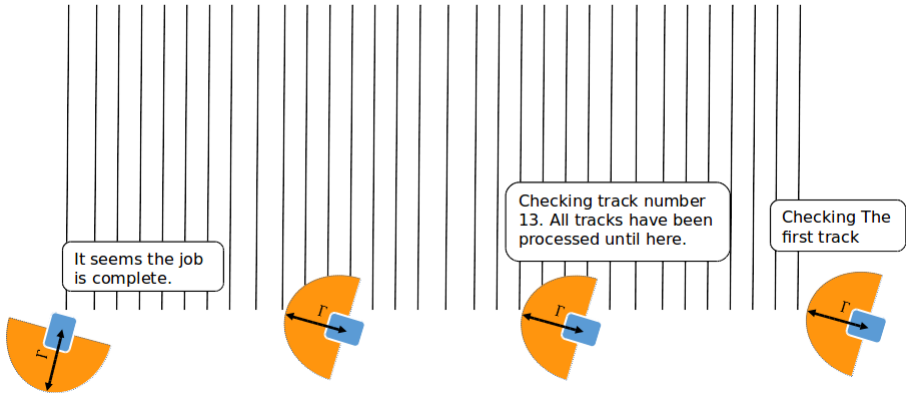


Figure 4.1: The ploughing interaction method cannot be applied to spraying or seeding since the furrows are already created.

In order to use the same cooperative algorithm, the considered task needs to have permanent and detectable effects on the soil which are different than the one that already exists. Let's assume a team of robots in which individuals are equipped with appropriate sensors to determine the level of chemical on certain locations. As mentioned in the initial condition in chapter 3, the robots are assumed to be scattered around the field, and thus the robots will access the field at different instances of time. If so, it is possible that by the time a robot is analysing a location to determine the level of chemicals, it concludes that the location has not been yet sprayed as the chemicals are evaporated. As a result, that particular location will be sprayed more than once (see Figure 4.2).

The tasks including spraying (the sprayed chemicals could be evaporated and it becomes untraceable) and seeding (the seeds might be covered with soil) create changes which are either temporary or time-consuming to detect.This creates another level of failure for each task as multiple robots could spray or seed certain area of the

field more than once.

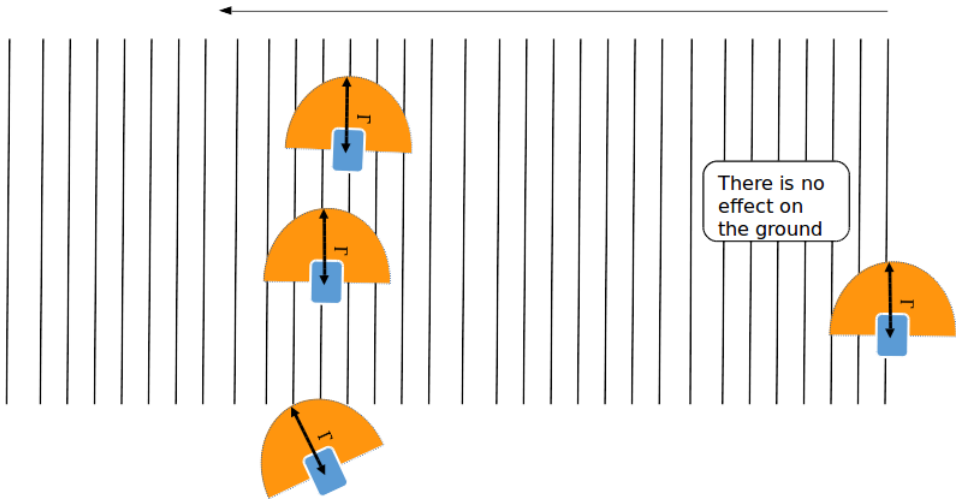


Figure 4.2: Any mistake in detecting the chemicals will result in excessive spraying.

It could be concluded that the solutions presented in the chapter 3 is only valid for tasks by which there will be permanent and detectable changes in the environment, and therefore it is inappropriate for tasks such as spraying and seeding.

One obvious solution is to divide the field into smaller regions and assign each region to a robot during design. With this solution, task allocation is carried out really safely and fast as excessive spraying is prevented. However, since robots have no information about the status of other participating robots, and since there are no explicit forms of communication the spraying will start asynchronously and in an uncoordinated fashion. As a result, even if the direction of execution is unified (i.e. robot spray the field in the same direction), it will be possible that two robots process two consecutive tracks from opposite directions. As a result, robots will reach congestions in the middle of the field. This is undesirable since robots cannot navigate in other directions as there are crops in the environment.

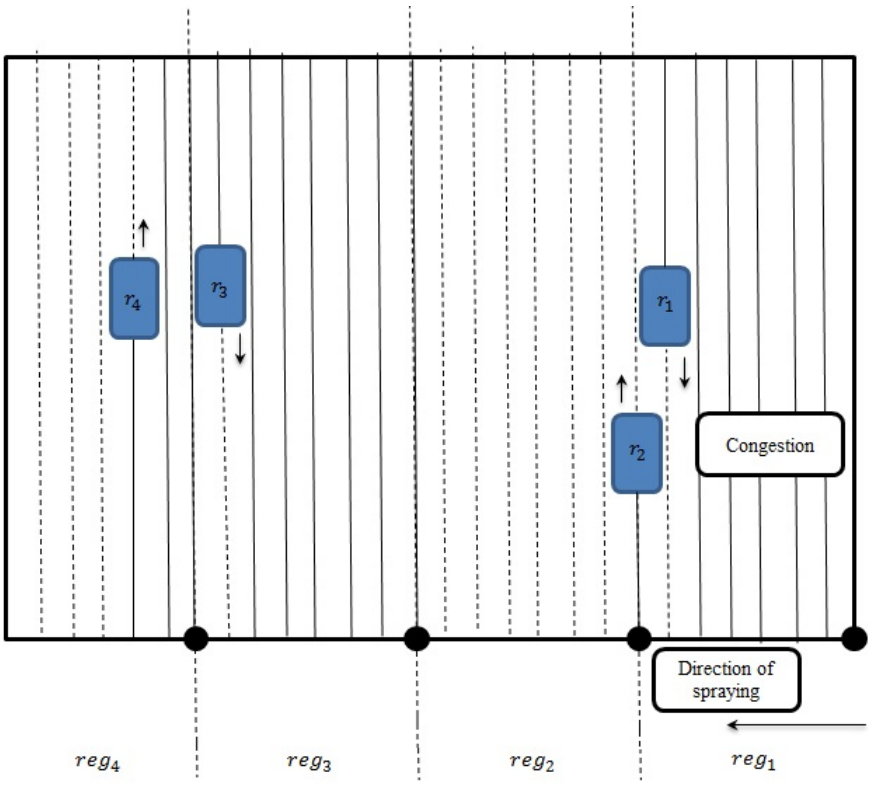


Figure 4.3: With static task allocation, congestions in the middle of the field are inevitable as spraying is uncoordinated.

These points are the motivations to investigate a new cooperative behaviour for a large team of robots by which a large field can be sprayed at different stages of cultivation. Although the targeted task is spraying, the provided solution should be applicable for similar tasks including seeding, re-seeding, seed mapping, and etc.

4.2 Spraying Analysis

Spraying is the process of dispensing Plant Protection Products (PPP) on the crop at different stages during cultivation. PPP includes herbicides, pesticides, fungicides, and growth fertilizers. According to Faivre et al. (in 2008) it is common to mix PPPs with water pumped through the irrigation system. For example, liquid fertiliser and/or insecticides can be drawn into the stream of water which is pumped from a water source such as a river or well. Proper application of the chemicals allows the crops to be grown with a bit more certainty since nutrient problems and/or insect infestations can be addressed while the crop is growing (Faivre et al., 2008).

The definition of spraying also encompasses water irrigation. The earliest archaeological evidence of irrigation in farming dates to about 6000 B.C. while the earliest pictorial representation of irrigation is from Egypt around 3100 B.C (Sojka et al., 2002).

Traditional irrigation methods vary from using natural flood cycles of the local rivers, in which the natural river stream was guided to the furrows (or crop rows) throughout a field to direct moisture to the plants therein, to trickle or drip irrigation methods, in which a small amount of water is applied to the plants to reduce evaporation of the water (Faivre et al., 2008). When high-pressure delivery systems became available, spray irrigation became popular because of the distance that it could cover.

The spray irrigation may additionally utilise machinery that relocates the spray nozzles throughout different portions of the field in a controlled manner. Conventionally, a tractor with a spraying unit is driven through the field and the PPPs are gradually dispensed on the crop. The task of spraying can be initiated from any point in the field and the tracks can be sprayed in any order (see Figure 4.4).

Figure 4.4: Spraying unit is driven on the tracks while PPPs are gradually dispensed on the crop (AAPlus, 2017).

Spraying is distinguished from other agricultural tasks in that of redundancy of processing. In other agricultural tasks (e.g. ploughing, seeding, and harvesting), even though redundancy in processing (that is processing a point in the field more than once) increases the cost of execution, the final result is still acceptable. In spraying, any location in the field has to be processed only once, since excessive PPPs dispensing will destroy the crop. Moreover, in spraying, the direction of field processing is fixed. The sprayer unit is allowed to navigate through the field via gaps between the crop rows referred to as tracks (since spraying is carried in various stages of cultivation, furrows may or may not exist, therefore they are referred to as tracks). Any other motions or manoeuvres are prohibited since the crop will be run over.

4.2.1 Single Robotic Sprayer

In the recent years, automated spraying system has been improved significantly, and today there are variety of spraying units from time-based static sprayers to autonomous mobile sprayers.

One of the well-known autonomous spraying units is Irrigation Robot developed by John Deer (Figure 4.5). The Irrigation Robot can move in two directions and spray the field uniformly. Although the Irrigation robot can be extended or reduced in size for different field sizes (Faivre et al., 2008), the field has to be relatively flat with minimal rough terrain. Besides, the accuracy of the spraying unit is relatively low since the dispense of the materials is from a long distance.

Figure 4.5: Irrigation Robot (tarmakbir, 2017)

Accuracy in spraying motivated various researchers to apply mobile robots in this field. For example, Aarhus University in Denmark developed a semi-autonomous mobile robot for weed removing (see Figure 4.6). Using an ECODAN camera, GPS system, and a gyro sensor, the robot navigates the entire field. While in the field, the robot continuously captures pictures of the ground and sprays only the detected weed. The robot can detect up to 25 different types of weed, and as a result, it saves the usage of herbicides by 75 percent (Sujaritha et al., 2016). Using a vision system to detect and distinguish the existing weed from the crop is a popular approach in

weed control robots. (Bakker, 2009), (Choi et al., 2015), (Sabanci and Aydin, 2017) and (Aiswarya et al., 2016) are few examples of vision based spray robot.

Figure 4.6: Hortibot made by Aarhus University in Denmark

Even though these robots are very soil friendly, they are considerably slow (Sujaritha et al., 2016). One way to reduce time in this approach is to increase the number of participating spray robots in the field. In the past few years, there have been very few attempts. Hansen et al. (in 2013) designed and implemented a cooperative structure for a team of two robots to spray herbicides on a large field. The described team consists of a UAS (Unmanned Aerial System) and a UGV (Unmanned Ground Vehicle) along with a Task Manager. The job of the task manager is to decompose the overall task in such a way that the UGV and the UAS perform the execution in the fastest and least resource-demanding fashion. The task manager continuously is communicating with the participating units, thus its success is prone to success in communication. Again, there has not been a cooperative system in which robots rely only on their local information.

4.3 Cooperative Spraying: Design Description

In this section, the proposed cooperative spraying is described by first describing the process of task allocation.

In spraying, task allocation has to guarantee that each location in the field is visited only once. In addition, task allocation has to be carried out in real-time, and only the local information of the robots is available. Therefore, a mechanism is required by which robots are informed about the state of the task. Note that this information cannot be conveyed with explicit forms of communication.

The proposed strategy is to divide the field into regions (while each region consists of a few tracks), with the aim to process each region by only one robot. For this, the robots have to identify unattended regions and claim their share of task. Each robot claims processing a region by occupying a particular location, hereafter referred to as checkpoint, whose locations are common knowledge among robots. In here, checkpoints are set to be the last track of each region. Spraying a region starts from this location and tracks are processed consecutively to the first track in the region. There is no restriction on which track is selected as a checkpoint or the direction of region processing. However, the direction of region execution has to be unified. In other words, spraying from first to last or last to first.

Note: if a region consists of tracks between track number n and $n+k-1$, k is the number of tracks on the region, $n+k-1$ is the last track and n is the first track in that region.

Therefore, robots have to check each checkpoint to see if the region is occupied. If an unoccupied location is found, the robot proceeds and occupies the location (see Fig. 4.7). Once occupied, the robots should not initiate the spraying to assure that all other robots are informed about the state of that region. All robots have

to standby at the occupied checkpoint except the last robot. Since there is no other unattended region, the task can be initiated. During this process, robots keep count of the number of visited (with occupied status) checkpoints. Knowing the total number of checkpoints, one can determine whether the current investigating checkpoint location is the last checkpoint.

.

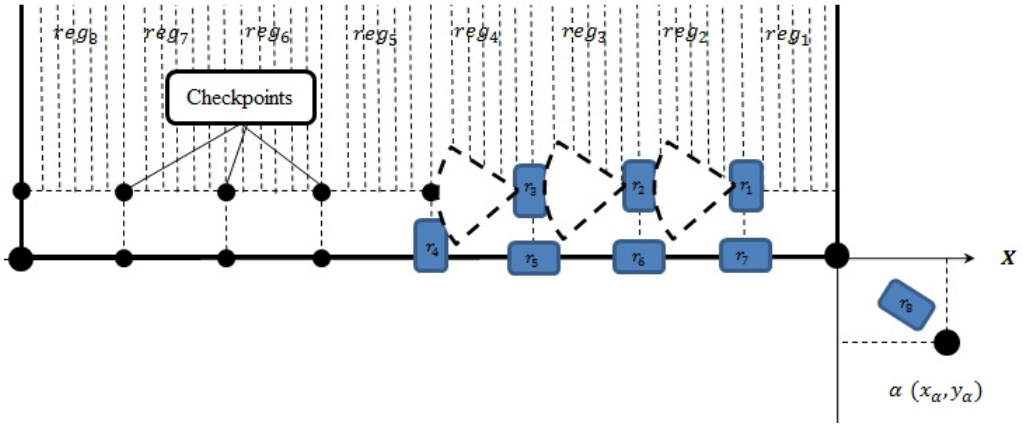


Figure 4.7: A team of 8 robots are performing task allocation. r_1, r_2, and r_3 have found unoccupied checkpoints. In the meantime, other robots are examining every checkpoint.

Once the task allocation is completed, other robots have to be informed otherwise they will never start spraying the field. To solve this, when a robot occupies a checkpoint, it poses itself in a way that it continuously monitors the next checkpoint. The last robot does not need to comply with this, and instead, it starts the task right after it occupies the last checkpoint. r_{n-1} which is contentiously monitoring the behaviour of r_n, identifies that r_n is appears and disappears from its field of view. This is the signal which indicates task start. Thus, r_{n-1} starts spraying once r_n is no longer detected. With this, robots start spraying one after another, and the first robot will be the last to start spraying(see Fig. 4.8).

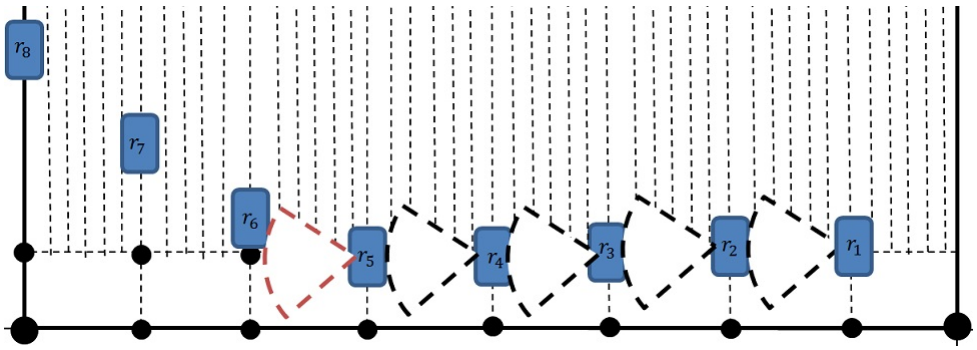


Figure 4.8: Task initiation stage for a team of 8 robots; Spraying task starts whenever r_8 reaches checkpoint on the last region. r_7 starts spraying once it perceives r_8 decision.

4.3.1 Task Partitioning Analysis

As mentioned, in the proposed approach the field should be divided into regions each of which consists of a set of adjacent tracks. Also, since regions are divided among the participating robots, the number of regions should be equal to the number of robots (or the team size). Therefore, if there are n robots and K tracks in the field, the number of tracks in each region, k_i, can be determined as follows:

$$k_i = \begin{cases} \lceil \frac{K}{n} \rceil & \text{if } i \leq K \mod n \\ \lfloor \frac{K}{n} \rfloor & \text{if } i > K \mod n \end{cases} \tag{4.1}$$

In here, $\lceil \frac{K}{n} \rceil$ represents the smallest following integer, and $\lfloor \frac{K}{n} \rfloor$ represents the largest previous integer.

Next, it is important to identify the track numbers within each region. Robots require this to locate the checkpoints. Let's assume that TR is the set of track numbers in the field, $TR = \{tr_l | l \in \{1, 2, ..., K\}\}$, and R is the set of robots, $R = \{r_i | i \in \{1, 2, ..., n\}\}$, then track numbers allocated to r_i, can be determined as

follows:

$$G_i = \{tr_j | j \in \{m+1, m+2, ..., m+k_i\}\} \tag{4.2}$$

Where m is the last track number assigned to the previous robots, and it can be calculated as follows:

$$m = \sum_{j=1}^{i-1} k_j \tag{4.3}$$

4.3.2 Task Allocation Analysis

The proposed task allocation mechanism consists of redundant checkpoint analysis. As the number of robots increases, the number of locations that the robot has to check also increases. To determine the time required for the team to complete task allocation, let's first define the duration of task allocation for a robot.

Definition 1. *Task allocation duration for a robot is the period from initial position until the time that the robot detects an unoccupied checkpoint.*

With this definition, time analysis becomes complex since robots are initially scattered around the field. To simplify estimation, let's assume that robots have formed a queue behind a location outside of the field. This location is referred to as *alpha*. Before robots access the first checkpoint, they first have to access *alpha*. However, the distance that a robot has to travel to reach *alpha* depends on the robot's position in the queue. The length of the queue for each robot is $(\lambda + \epsilon)(i + 1)$ meters. In here, λ is the length of a robot, and ϵ(*epsilon*) is the minimum distance between two consecutive robots (see Figure 4.9).

Once a robot reaches *alpha*, regardless of their position in the line, it has to analyse the first checkpoint, and hence all robots have to travel a fixed distance between *alpha* and the first checkpoint, d_{α,l_1}. Also, except the first robot, other robots have

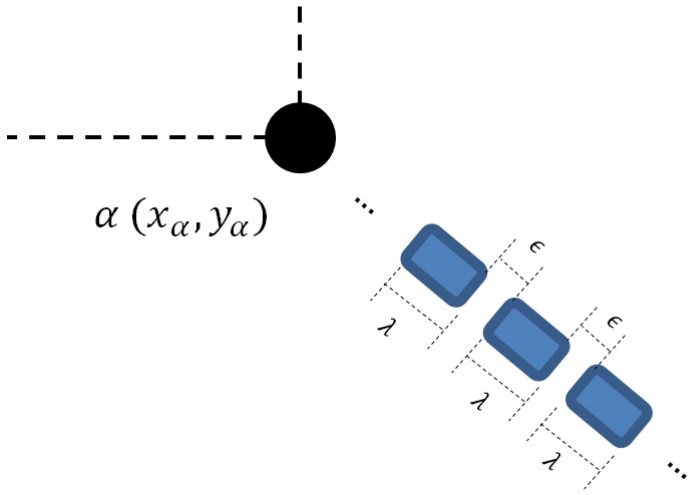


Figure 4.9: Illustration of robots queueing for accessing α. Robots have the same length, λ, and the distances between robots, ϵ, are equal.

to travel to other checkpoints, d_{l_{j-1},l_j}. For example, r_2 has to travel distance between the first and second checkpoint to reach its destination.

Once a robot reaches a checkpoint, it takes a period of time to draw a conclusion on the status of the checkpoint. This period is denoted by τ and it will be propagated in the queue since robots have to wait for the path to be cleared. The total delay for a robot is $(2i-1)\tau$.

This is easy to see. Let's consider a team of three robots which are lined up behind a point outside the field (α)(Figure 4.10).i). r_1 analyses only the first checkpoint, hence spends only a τ seconds. Consequently, the rest of the team wait τ seconds as well (Figure (4.10).ii). Once the path is cleared for r_2, it will analyse checkpoint one and as a result another τ seconds of delay is propagated to r_3 (Figure (4.10).iii). r_2 resumes its analysis to checkpoint two and spend another τ seconds on checkpoint two. During this period, r_2 creates no effect on the rest of the team since by this time, r_3 is already performing analysis on the first checkpoint (Figure (4.10).iv). r_3 resumes checkpoint analysis on checkpoint two (.Figure (4.10).v) and the last

checkpoint (Figure (4.10).vi).

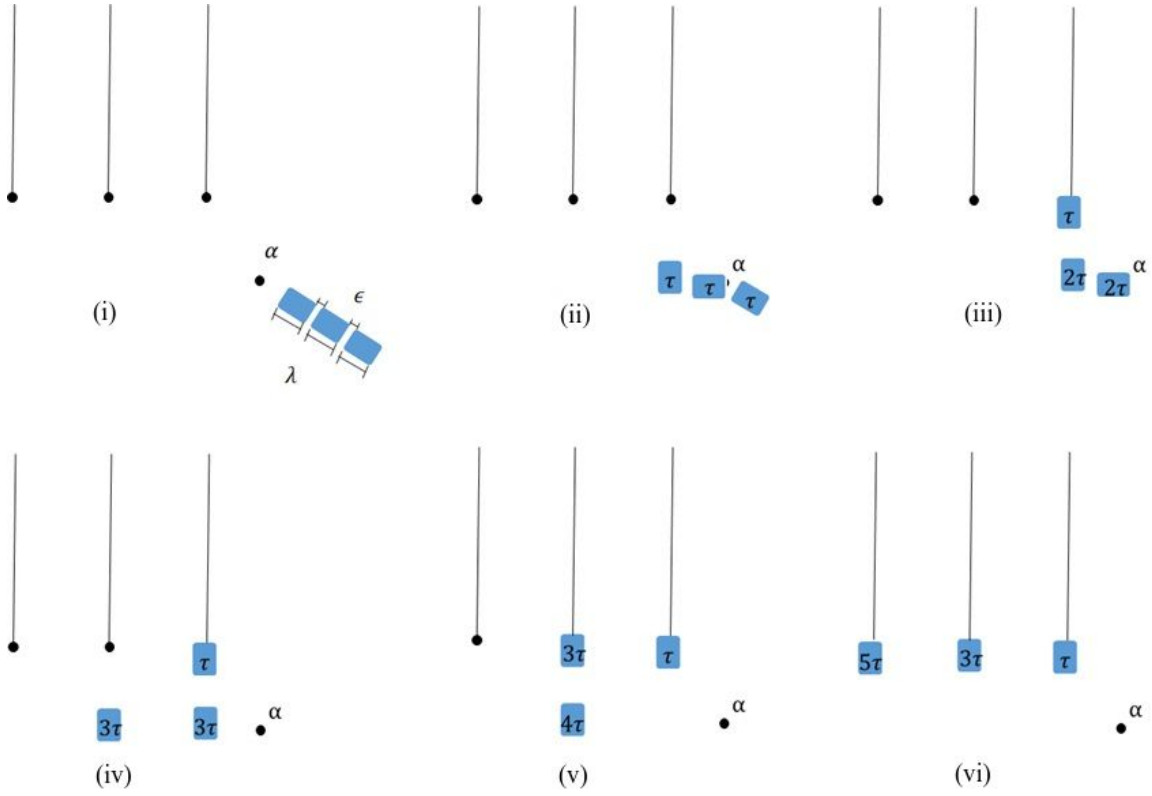


Figure 4.10: The redundant checkpoint analysis propagates delay in the team. (i) robots are in a queue behind α. (ii) r_1 is processing the first checkpoint, all other robots have to wait until r_1 completed its analysis. (iii) r_2 is analysing the first checkpoint, another delay is affecting the unassigned robots. (vi) r_2 is processing the second checkpoint, however there is no effect on r_3 as it is processing the first checkpoint. (v) r_3 is checking the second checkpoint. (iv) r_3 also needs to check the last checkpoint.

If the velocities of all robots are equal and robots move with constant speed, task allocation duration for a robot (r_i) can be evaluated as follows:

$$t_{ta_i} = \frac{d_{\alpha l_1}}{v} + (2i-1)\tau + \frac{(\lambda+\epsilon)(i-1)}{v} + \sum_{j=2}^{i} \frac{d_{l_{j-1}l_j}}{v} \tag{4.4}$$

4.3.3 Task Initiation Analysis

Task allocation for a robot is completed when it identifies the first unoccupied checkpoint, but the task is not yet initiated since the robot has to assure that all other robots are informed about the status of the occupied region. Before any further analysis, let's define task initiation duration.

Definition 2. *Task initiation duration for a robot is the period that a robot has to remain at a checkpoint until the initiation signal is detected.*

Since spraying will not start for all robots until the last robot has occupied the last region, the task initiation time for a robot will be affected by the last robots task allocation time. However, this impact is partially compensated by the robot's task allocation period. With this, if τ_{init} is the constant time to perceive task initiation event (that is appearing and disappearing of the next region robot), the standby period for a robot before it starts its task is evaluated from the following:

$$t_{st_i} = t_{ta_n} - t_{ta_i} + \tau_{init}(n-1) \tag{4.5}$$

With this strategy, r_1 is the first robot that completes the task allocation, but it is the last robot to initiate spraying. Therefore, (4.5) can also be expressed as follows:

$$t_{st} = t_{st_1} = t_{ta_n} - t_{ta_1} + \tau_{init}(n-1)$$

In here, t_{st} refers to total spraying execution time, and t_{st_1} is the first robot spraying time.

4.3.4 Spraying Time Analysis

Right after the task is initiated, robots start spraying. Spraying is a two-dimensional navigation task which is performed asynchronously (i.e. during spraying robots only

concentrate on the allocated region. Plus they do not require any information from other participating robots during this stage). If the velocity of a robot is denoted by v, and the length of a track is denoted by l_p, then spraying time for a robot after the task initiation can be evaluated as follows:

$$t_{spraying_i} = \frac{1}{v}(k_i l_p + (k_i - 1)d_f) \tag{4.6}$$

In here, $t_{spraying_i}$ refers to the spraying time for robot i, d_f corresponds to the distance between two consecutive tracks. Bear in mind that a robot has to repeat spraying as many as k_i times, obtained from (4.1). In addition, as part of spraying, a robot also has to switch between tracks for $k_i - 1$ times.

The total system execution time including all three steps (task allocation, task initiation and spraying) is the sum of maximum in each task. The field is partitioned from the first regions, therefore it is safe to conclude that the first region always has the highest number of rows to cover and the maximum spraying time (t_{s_1}). It is the last robot that has the longest task allocation time t_{ta_n}, but it is the first robot that has the longest task initiation time t_{ts_1}.

$$t = t_{s_1} + t_{ta_n} + t_{ts_1} \tag{4.7}$$

4.3.5 Design Limitations

The proposed strategy promises that a group of robots can spray a large field cooperatively relying only on their local sensing. However, there are few limitations in the system, and this section is dedicated to review them.

Extreme Sensitivity: One main limitation of the designed system is that in order for the task to begin, all checkpoints have to be occupied. But what if one or more

robots fail during the process of task allocation? As a result, (since the number of robots and number of regions are equal) the task will not begin and the field will never be sprayed (see Figure 4.11).

This flaw could simply be compensated by increasing the number of participating robots. But how far can the system tolerate this increase?

In particular, if R_m represents the number of participating robots, and Reg_n represents the number of available regions:

$if Reg_n < R_m < 2 * Reg_n$ the field will be sprayed once, but the system will be inefficient as there will be robots which are not participating (see Figure 4.12).

$if R_m \in \{2 * Reg_n, 3 * Reg_n, 4 * Reg_n, ...\} \rightarrow$ the field will be sprayed more than once (see Figure 4.13).

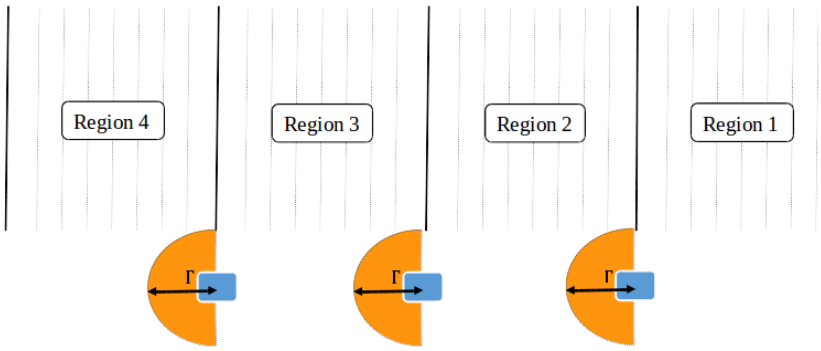


Figure 4.11: $R_n < Reg_n$: The spraying task will never initiated

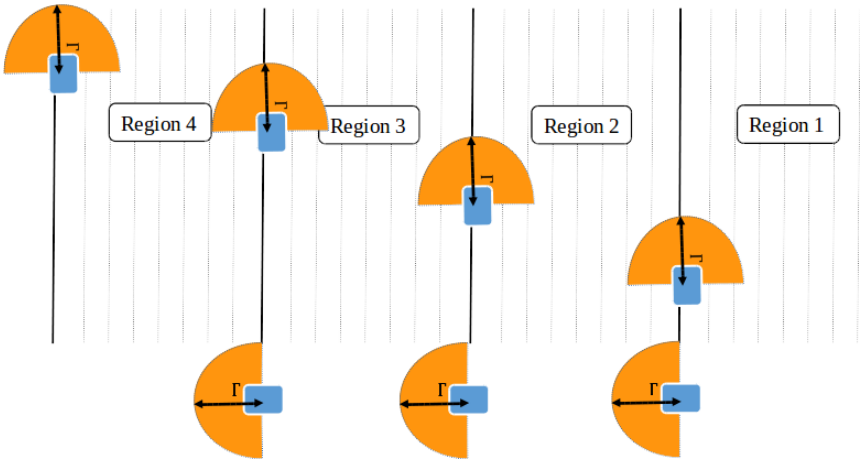


Figure 4.12: $Reg_n < R_n < 2 * Reg_n$: The spraying begins, and there are robots which will never participate in spraying, but they have to perform redundant task allocation stage.

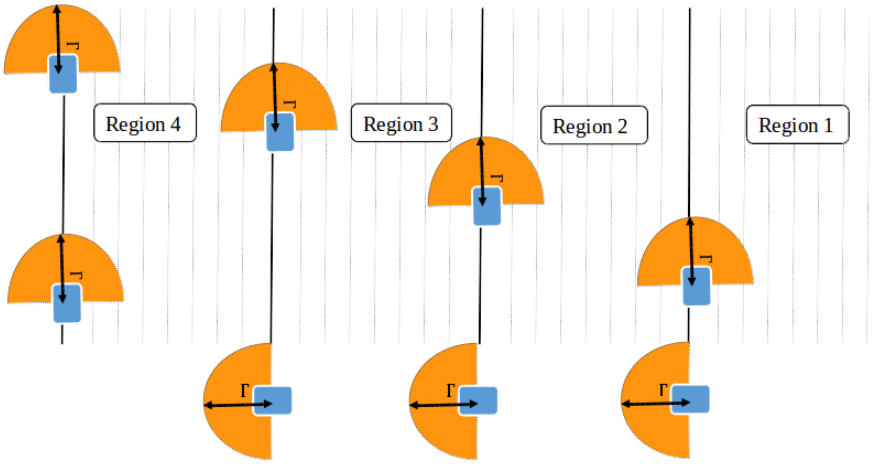


Figure 4.13: $R_n \in \{2 * Reg_n, 3 * Reg_n, 4 * Reg_n, ...\}$: The field will be sprayed more than once.

Limited Range of Perception: Robots rely on their local sensing for task allocation particularly during task initiation period (see Figure 4.8). However, robots can perceive only a limited range. Let's denote this range by Γ. If the distance between two consecutive checkpoints (i.e. the distance between two consecutive robots during task initiation period) is greater than Γ, the robots will not be able to detect the initiation signal as they are not able to observe the behaviour of the adjacent

robot. As a result, only the last region will be sprayed because the last robot is not required to observe any other robots. Figure 4.14 demonstrates this situation.

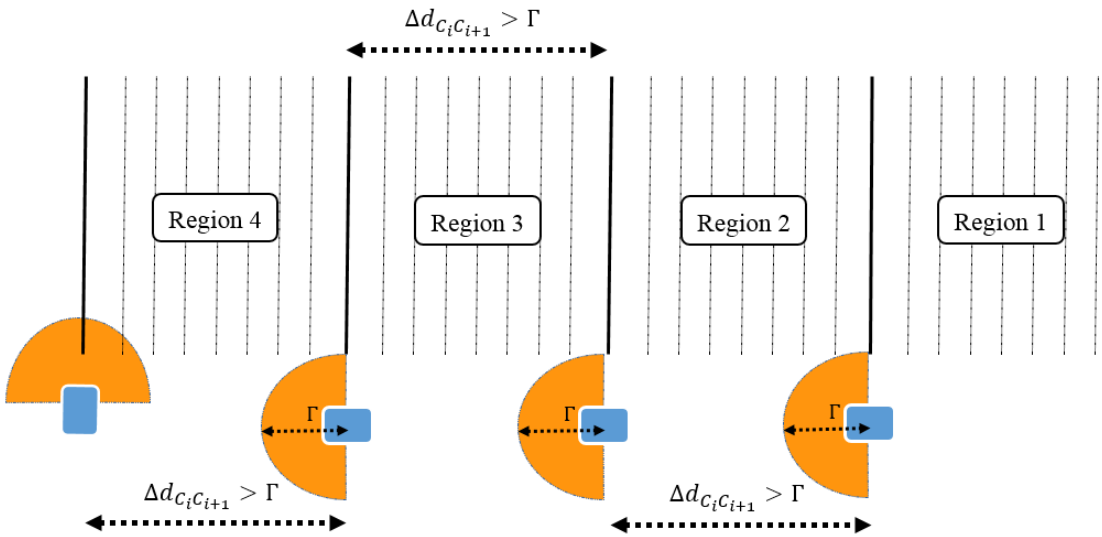


Figure 4.14: If the distance between two consecutive checkpoints is greater than the detection range of the robot, it cannot monitor the behaviour of the adjacent robot. Thus, the task will never begin for that particular robot.

Since the maximum distance between two consecutive robots cannot exceed the detection range of the robots, it is possible to determine the minimum number of regions which are required to prevent spraying a field.

$$n_{min} = \lfloor \frac{W}{\Gamma} \rfloor \tag{4.8}$$

Physical Dimension Limitation: Robots have physical dimensions ($\lambda \times \zeta$), and similar to ploughing they need free space to perform track transitioning after one cycle of spraying, and for actual spraying in opposite direction. This will limit how close the checkpoint can be appointed. The minimum required distance between two consecutive regions is $\lambda + \epsilon$. In here, ϵ is the threshold collision distance between two robots. The minimum distance defines the maximum number of participating

robot:

$$n_{max} = \lfloor \frac{W}{\lambda + \epsilon} \rfloor \tag{4.9}$$

4.4 Implementation and Testing

In this section, we present both numerical results based on mathematical consideration and the results obtained from the Stage simulation environment.

4.4.1 Mathematical Results

Since the allowed number of participating robots depends on the dimension of the field, first the boundaries of the team sizes have to be identified. In here, a few parameters about the environment have to be fixed.

We assumed that there are 50 to 250 tracks available in the field and the distance between two consecutive tracks is 0.2(m), and the length of each track (l_p) is 20(m). Robots have equal dimensions with length (λ) equal to 0.5(m). The threshold distance (ϵ) between two robots is set to be 0.3(m), and all robots are assumed to move at constant velocity equal to 0.5(m/s). The checkpoint analysis duration (τ) and robot behaviour analysis duration (τ_{init}) is fixed to 5(s).

From equations (4.8) and (4.9), it is possible to identify the maximum and the minimum number of robots allowed in various field sizes, since $W = Kd_f$. Figure (4.15) demonstrates the results of Matlab simulation for various field sizes and their appropriate field team sizes. In Figure (4.15) the red line indicates the maximum team size and the blue line indicates the minimum number of participating robots. Note: The definition of a team in a multi robot system fixes the minimum number of robots to be greater than three ($n_{min} \geq 3$).

For a field with hundred tracks ($K = 100$), the number of participating robots could

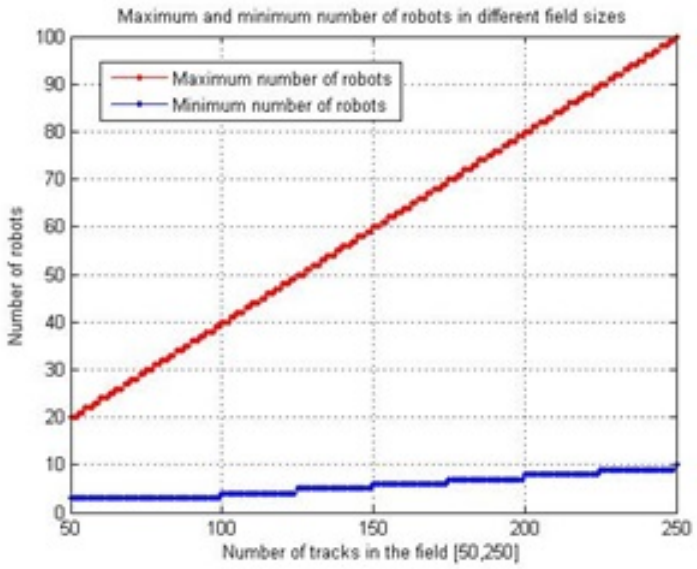


Figure 4.15: Number of participating robots against number of tracks in the field

vary between four and forty ($n \in [4, 40]$). Consequently, task allocation and task initiation time can be predicted (see Fig 4.16).

As the number of participating robots increases, the duration for both task allocation and task initiation increases by which the total execution time increases. In a field with 100 tracks, $K = 100$, if appropriate team sizes are applied ($n \in [4, 40]$), the total execution time opposes with what is expected from the nature of a multi robot system (see Fig.4.16). It could be concluded that the applied strategy is not efficient enough to receive positive effects from an increase in the number of robots.

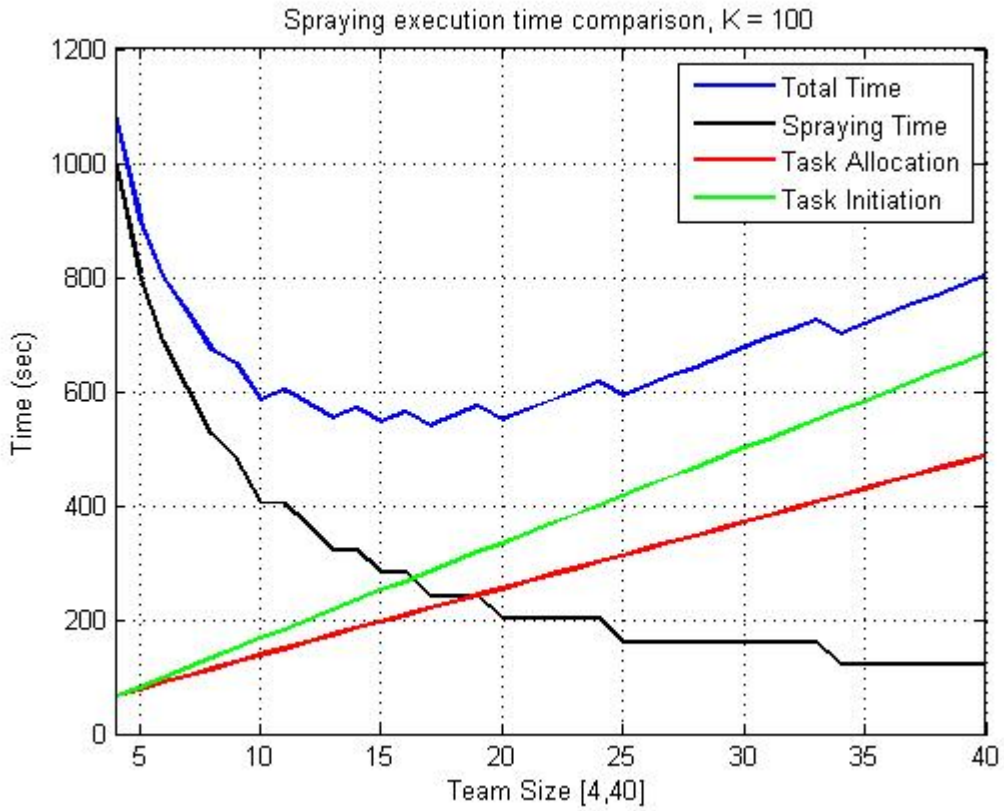


Figure 4.16: Region-based Approach Time Analysis. The resulted graph is obtained via Matlab

4.4.2 Simulation Results

On the other hand, a series of simulations were conducted in Stage simulation environment in ROS (Robot Operating System). Similar to the simulation environment described in Section 3.8.3, each robot is equipped with a model of a Hokuyo Laser Range Finder, and a fixed camera which both are placed at the front of the robot. The current location of the robot in the global frame of coordinates is provided by the simulation environment. However, during the trials, no robot is aware of the position of other robots in the team.

Behaviours on each robot are controlled by collaboration between three C++ modules (see Figure 4.19): (1) Task Handler, (2) Reach Point, and (3) Camera Analyser. All three modules are communicating via ROS specific messages via namedPipes.

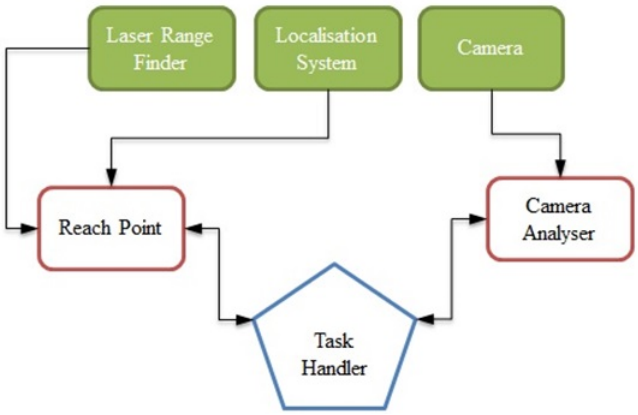


Figure 4.17: Spraying Software Overview

The **Task Handler** is responsible to trigger specific behaviours in the robot: setting a new target for Reach Point module, activating Camera Analyser, analysing the field, and executing the cooperative procedures. Figure 4.18 demonstrates Task Handler flowchart, and Appendix J contains C++ implementation.

Reach Point is responsible to guide the robot to the requested coordinate in a collision-free manner. As explained in chapter 3, Artificial Potential Field is used for navigation. The exact same module can be reused in here.

Camera Analyser is responsible for (a) determining if checkpoints are occupied, and (b) determining when to initiate the task. This can solely be carried out if other robots can be distinguished from all other objects in the environment. In here, robots are assumed to have a different colour than other objects in the environment. Therefore, the camera analyser performs colour detection upon the task handler request.

(Janani, 2015c) [1] contains a link to the video for the simulation of the cooperative spraying.

[1]Spraying Simulation Video Link:

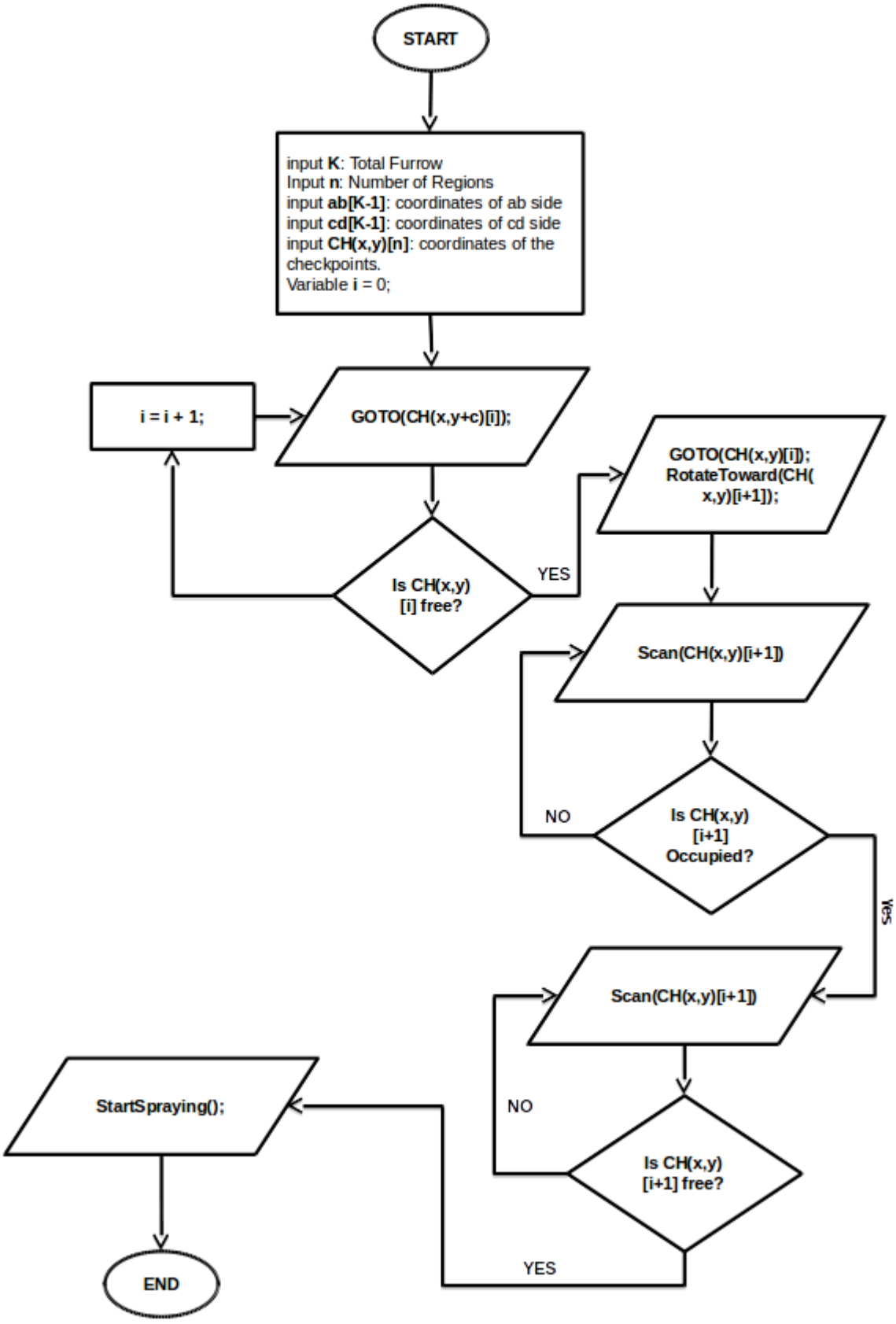


Figure 4.18: Region-based Task Handler Flowchart

Trials conducted for a field with 51 tracks (K = 51), and various team sizes were deployed ($n \in [3, 10]$). In all team sizes, robots performed task allocation and task initiation successfully. During simulation, position and time of robots are recorded. Data recording for a robot initiates when the robot passes α, and it stops as soon as the robot exits the margin of the field. The maximum execution time then is plotted and compared with the equivalent corresponding numerical prediction (see Figure 4.19). It can be seen that there is a slight difference between Stage simulation results and the numerical results. This difference is due to the field exiting duration which has not been considered in the mathematical description.

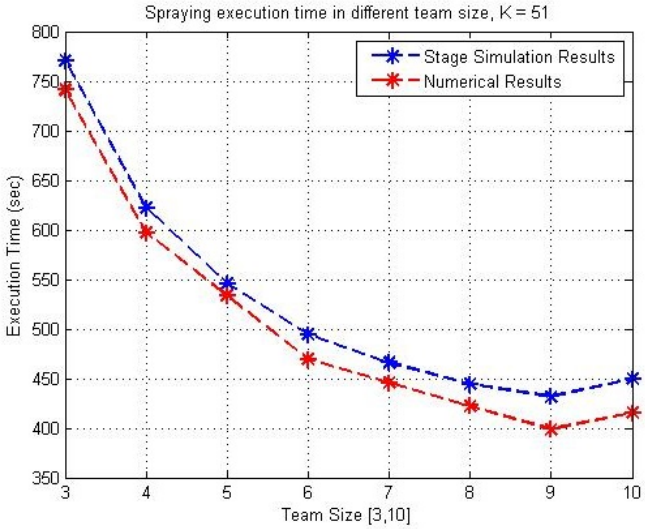


Figure 4.19: Comparison between results collected during simulation and numerical visualisation.

4.5 System Optimisation

The proposed cooperative method relies solely on local sensing of the robots. With local sensing comes certain limitations. These limitations have impacts on the system performance. In here, we review these impacts and implement further actions to overcome these restrictions.

4.5.1 Dynamic vs Static Checkpoints

The proposed method is confined by the detection range of the robots. This in return limits the width of regions that a fixed number of robots can spray. Let's consider a large field ($100m \times 100m$). Let's also assume that there are only 5 participating robots. According to subsection 4.3.5, this team size is inappropriate for this field and the field cannot be sprayed. In this section, the aim is to demonstrate a simple solution with which a small team can spray a large field.

The source of this issue is the position of consecutive checkpoints. With the current solution, the checkpoints are appointed according to the last track of each region. As the allocated regions are widened the distance between two consecutive checkpoints increases. In order to use a small number of robots for a large field, the distance between two consecutive checkpoints has to be within the robot's range of vicinity (see Figure 4.20).

4.5.2 Optimum Team

Referring to figure 4.16, it is clear that by increasing the number of participating robots despite being within the permitted range, demonstrated in figure 4.15, the overall spraying time will not decrease. This is due to the presence of two increasing linear functions and one decreasing hyperbolic function in formula (4.7). Therefore

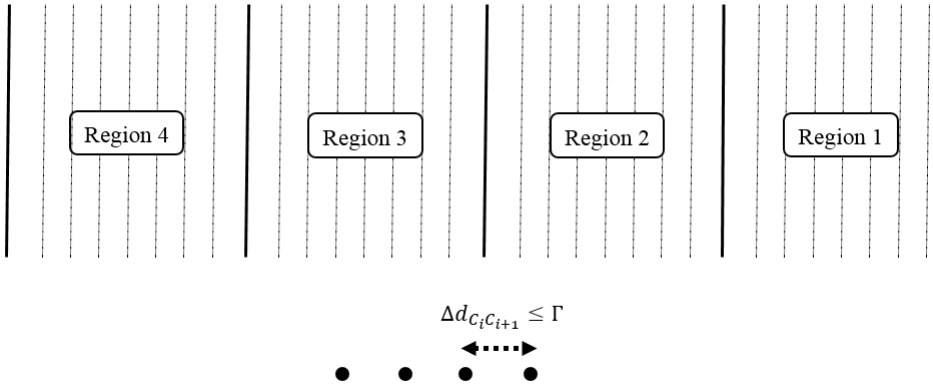


Figure 4.20: Check points are set outside of the field and the distance between them is reduced so that few robots are enabled to spray a large field.

the resulting function will have a global minimum. This is the optimal team size that could be deployed to the given field and it can be obtained as follows:

$$n_{opt} = \sqrt{\frac{K(l_p + d_f)}{2\tau v + \tau_{init} v + \lambda + \epsilon}} \tag{4.10}$$

In here, K is the number of tracks, l_p is length of the track, d_f is the distance between two consecutive tracks, τ is the image processing delay for checkpoint analysis, τ_{init} is task initiation image processing delay, λ is the length of the robot, and ϵ is the threshold collision distance between two consecutive robots. Derivation of 4.10 is provided in Appendix A.

The optimal number of robots corresponding to the minimum time from the start of time allocation and the completion of spraying depends on the parameters in the expression is 15.6. So the optimised number of robots is either 15 or 16.

4.5.3 Large Team and Fewer Checkpoints

As mentioned, the main target in task allocation is that all checkpoints have to be occupied. The proposed solution has limited flexibility. If a robot fails during this stage, the team will fail to initiate the task, and the field will never get sprayed. This

is due to the nature of the solution as there is no central controller to monitor robots behaviour, and there is no explicit communication between the robots to receive the latest updates. One way to reduce the probability of team failure is to increase the number of participating robots. However, the system is designed in a way that by the increase of the team size excessive spraying will occur if the number of participating robots is at least double the number of checkpoints ($N_{robots} \geq 2 * N_{checkpoints}$). When the first wave of robots start the task, the second wave of robots have no information about the status of the task, and hence the robots resume the process. This situation is demonstrated in Figure 4.13.

It is clear that some form of coordination between the robot that occupies the last checkpoint and other robots which have not occupied any checkpoints needs to prevent any excessive spraying. One way to coordinate them is to use the last checkpoint robot as an indication for others that their participation is no longer needed. Originally, the last checkpoint robot initiates the task right after last checkpoint occupation, but in this method, the last robot remains at its location until another robot is in place (see Figure 4.21). Once the robot, hereafter referred to as Patrol Robot, is in place, others can proceed with the rest of the task.

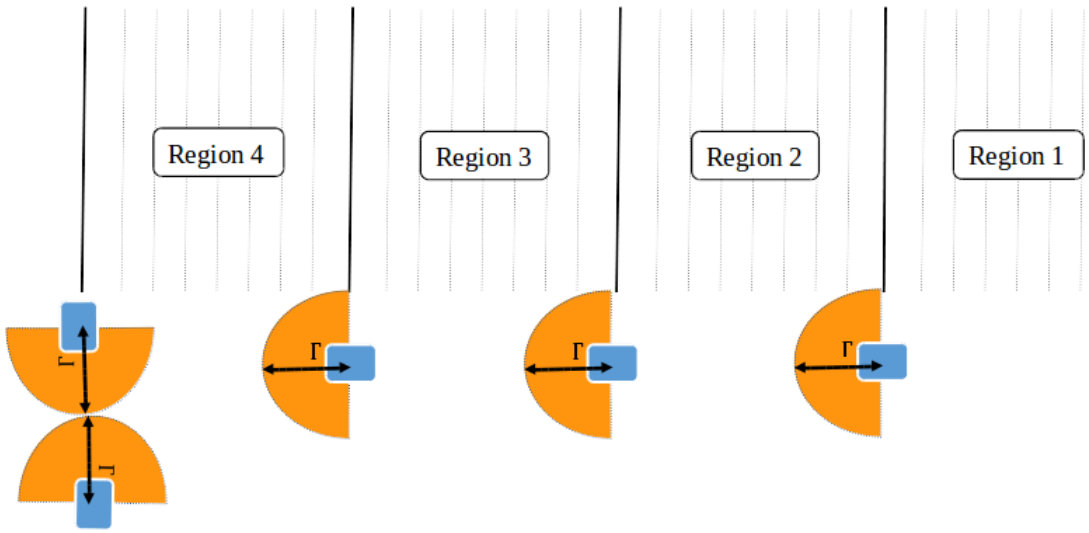


Figure 4.21: Last checkpoint robot does not start its task until Patrol Robot is in place.

The Patrol Robot, knowing that the task is fully partitioned and distributed, occupies the last checkpoint region upon task initiation with which any further execution will be prevented. As further waves of robots complete occupying the rest of the checkpoints, one by one, robots are notified that the last checkpoint is occupied earlier. If the first wave, the event perceived from the vision system indicating task initiation is Undetected-Detected-Undetected of the last checkpoint, however in this situation the event will be Detected-Detected-Detected meaning that the last checkpoint was occupied before the robot attendance (Figure 4.22). The conclusion is drawn that the task has started already and hence they return back. Upon this action, going home, the robot occupying the previous checkpoint perceive the direction by which the action is performed and consequently it concludes that the task is completed. This information is propagated down the line until the first checkpoint (Figure 4.23).

But what is the effect of this procedure on the overall execution time? With this additional procedure, as if another checkpoint is added to the collection. That is if

there are n number of regions, there need to be $n+1$ corresponding checkpoints. Therefore, task initiation required time obtained in 4.3.3 can be re-written as:

$$t_{st} = t_{st_1} = t_{ta_{n+1}} - t_{ta_1} + \tau_{init}(n) \tag{4.11}$$

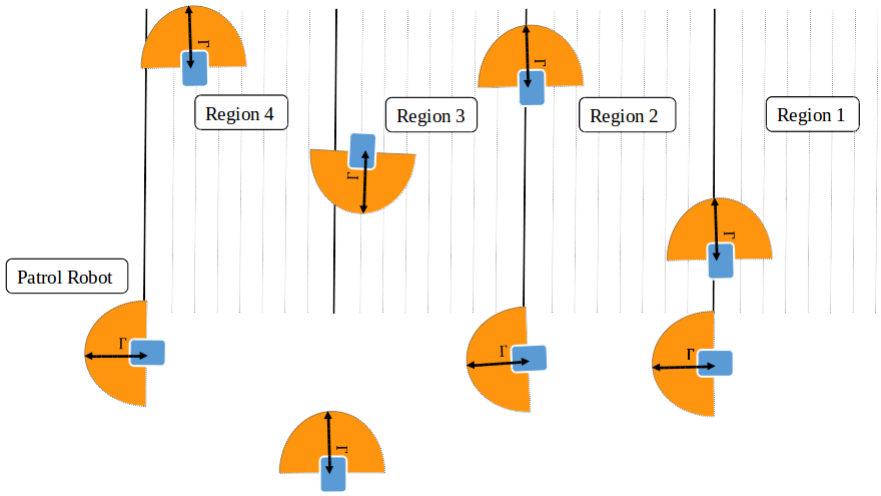


Figure 4.22: The presence of the Patrol Robot in the last checkpoint before the robot at $n-1$ checkpoint indicates that the field is already being processed and no further action is required.

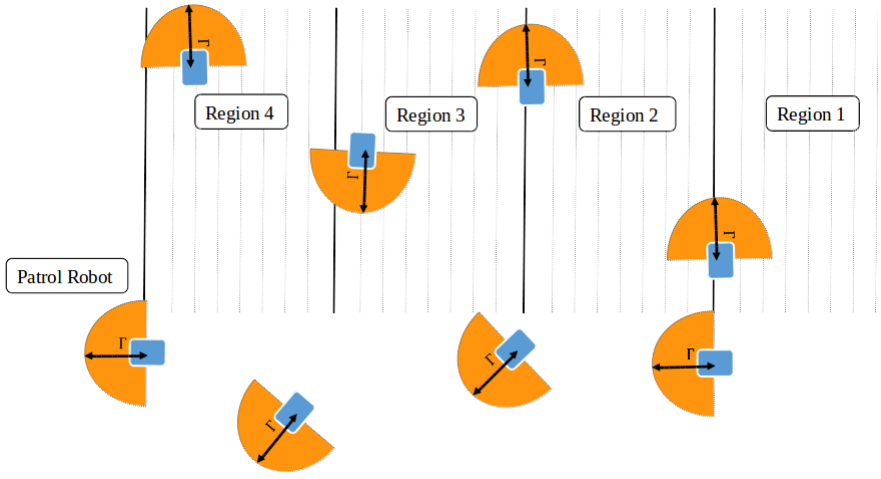


Figure 4.23: Upon detecting the presence of the Patrol Robot, other robots one by one leave the area.

Once spraying the field is completed, the Patrol Robot is informed to leave the

checkpoint by the participating robots passing from its vicinity (see Figure 4.24). With this modification, the robustness of the system will be increased as the system becomes independent of the number of participating robots.

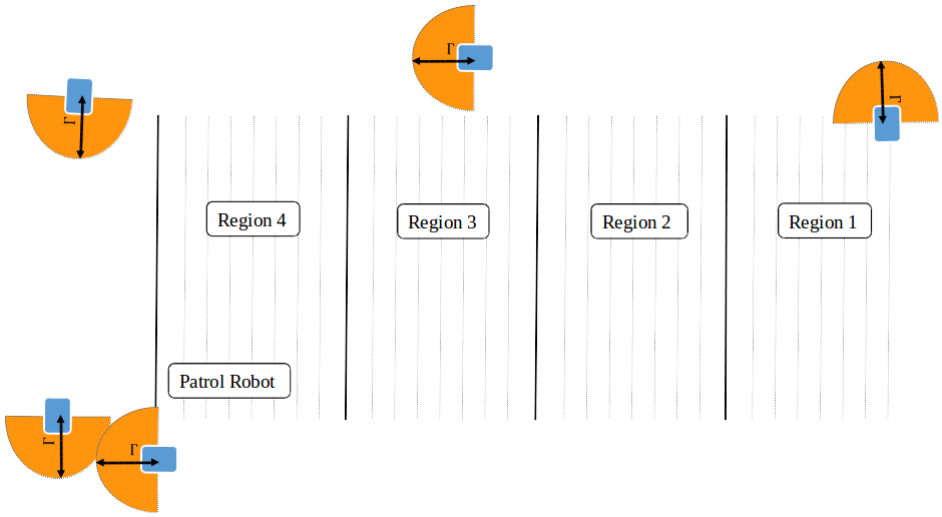


Figure 4.24: The movement of other robots within the vicinity of the Patrol Robot is an indication of task completion.

4.6 Conclusion

As mentioned, the restrictions defined by the task of spraying do not allow the cooperative approach designated to the task of ploughing to be applied for the task of spraying. Specifically, the method that robots interact with each other to obtain necessary information is only appropriate in ploughing since the execution result of ploughing does not exist in spraying.

The main idea of this project is to use the same robotic team for various open field related tasks including ploughing, spraying, harvesting, and etc. Therefore, another interaction and coordination method have to be investigated for spraying. Since explicit forms of communications do not exist in the robot, indirect methods are only considered. However, spraying has temporary impact on the environment, thus stigmergy-based interactions are inappropriate. Thus robots can only interact with each other by monitoring each others behaviour. This was the focus of this chapter: to answer cooperative-related questions including task allocation, and coordination with a specific form of indirect interaction.

In the proposed approach, the task allocation is carried out in real-time and during the execution time in which robots occupy designated locations referred to as checkpoints. While uncertain about their share of the task, the robots search for an empty checkpoint to occupy. The checkpoints are appointed during design period and all robots are aware of them. Once a checkpoint is occupied, the robot remains at the location to assure that no other robots will spray the region corresponding to that location, otherwise, excessive spraying of chemicals will ruin the crop. During this stage, they continuously monitor the adjacent checkpoint. The task will be initiated once the last checkpoint is occupied. The event of occupying and leaving the checkpoint by the last robot triggers the initiation task for the $n-1$ robot. This will

propagate down to all checkpoints and robots one after another initiate the task. While spraying, the interaction among robots will be reduced to simple collision avoidance as they need no other information to execute the rest of the task. The robots in the proposed solution face similar issues as in ploughing. This includes mainly spacial congestion and collision avoidance. The implemented solutions in ploughing can be transferred to spraying with minor to none modification.

4.7 Critiques and Future Work

Robots in this approach rely solely on their local sensing with which there will be certain limitations. The limitations are reviewed and appropriate actions are taken, and as a result, the robustness of the system is improved to a certain extent. However, there are still few points which result in total failure of the team.

Firstly, even though Patrol Robot prevents the overspray of the field and prevents the other robots to initiate the extra round of spraying, the proposed update still does not entail how this is prevented when the Patrol Robot has left the checkpoint. Robots are scattered around the field, and field accessing is asynchronous, thus one or more robots may access the field when the task is finished. Consequently, the field may be resprayed if enough robots are present.

Moreover, as robots do not interact with each other during the task of spraying, if a robot fails in the middle of the field, how would other robots be informed about this situation. Somehow, this information has to be propagated through the team so further actions can be taken.

These points have been identified as shortcomings of the proposed approach for cooperative spraying, and they are suggested as further research directions.

CHAPTER 5

Discussions, Conclusions and Future Works

In this chapter, a review of the research and the proposed approaches is provided 5.1. All proposed solutions in different aspect are compared in 5.2. The applicability of the proposed approaches and their outcomes in other fields of application is discussed in 5.3. And finally, the research is criticised, and the future directions of the research and conclusion are presented in 5.4.

5.1 Research Recap

The main goal of this research was to achieve the required cooperation using only the local information of each robot. In this way, task allocation and coordination among robots can be achieved independent of the existence of a central unit and communicating signal. However, the algorithm depends on the given task.

In open field farming, tasks are either *sequential* or *concurrent*. In sequential tasks, the field is processed in a particular and strict order, and any violation of the execution order cannot be tolerated and the results are considered failure. Ploughing is an example of sequential tasks. In furrow (narrow trench) and ridge (hill-top soil) type of ploughing, the plough creates a furrow by depositing to soil on a ridge. When creating the next furrow the soil should be deposited in the previous furrow. Violating the order will result in valleys (two consequent furrows) or crowns (two consequent ridges). This in return will cause irregularities in irrigation throughout

the process. On the other hand, in concurrent tasks, the field can be processed in any desired order. Spraying is categorised as concurrent.

In our field investigation, it was revealed that the task of ploughing is carried out using two types of ploughing mouldboard: reversible (with which the ploughing mould board can be rotated) and conventional (with which the direction of the ploughing mouldboard is fixed). Using conventional mouldboard, the field could be ploughed in one direction in order to maintain the mentioned pattern. However, with reversible mouldboard, the field could be ploughed from both ends of the field. This could benefit the ploughing cost as the ploughing units would require to travel less distances to cover the entire field.

With this in mind, first, two approaches were proposed for the task of ploughing (FIFO and LIFO) in which the robots were assumed to be equipped with reversible mouldboards. In both described approaches, the main idea was to carry out the majority of cooperation during the design time by performing offline task allocation: to allocate series of ploughing locations to each robot once robots found an order in the team. Knowing the number of participating robots, to find the order, robots move to the first ploughing location assuming they are the first robot in the order. With the use of the front-facing 2D-camera, they could identify the status of that location (whether it is ploughed or not). If it was ploughed, they move on to the next ploughing location and increment their order in the team. Once an unploughed location is identified, they would know their rank and could identify the sets of ploughing locations that is allocated to them.

Since the ploughing is carried out in both sides, robots could not plough the next location unless all robots have completed their ploughing. That is necessary to guarantee the required sequential ploughing pattern. This means that a delay from

a single robot will be propagated throughout the entire team. Even worse, if a robot fails during or before task execution, the entire system would fail.

With this newly learned point in mind, the self-organised approach was introduced. In this approach, the idea of benefiting from the reversible ploughing mouldboard and the knowledge of the participating robots are ignored. The task allocation is performed in real-time in which robots have to seek for a new ploughing location every time. In this way, if a robot fails after a round of ploughing or before even starting the ploughing the team can recover from this failure and complete the task. Although redundancy in the self-organising approach is higher, it is more robust toward failure of a single robot.

The self-organised approach for the task of ploughing works only because the process of task allocation can be carried out through changes made by other robots in the environment (i.e. robots can detect and resume the task which is done by others). However, this feature does not exist in other open farm tasks. Hence, the self-organised approach will fail if such feature does not exist. The task of spraying is an example of these tasks.

Therefore, for spraying, a new approach is proposed in which robots perform the process of task allocation relying only on monitoring each others behaviour. Spraying can be categorised as a concurrent task. This means that the field can be processed from multiple locations independently. Using this characteristic, in the proposed strategy, the field is divided into smaller manageable regions.

The strategy is to spray each region by a unique robot so that excessive spraying is prevented. Then, for each region, a representative location is designated outside of the field called checkpoints. The checkpoints are appointed during the design time and they are made known to all robots. The robots, one after another, examine

the checkpoints one by one to find the first unoccupied checkpoint. Once an unoccupied location is identified, the robot occupies it and position itself so that the very next checkpoint can be observed. In this way, the robots will inform others by their presence that the correspondent region is claimed, also they can monitor any activities of the adjacent neighbour. In theory, the robots have to remain in their position until all checkpoints are occupied. Once a robot finds the last checkpoint unoccupied, it proceeds but instead of remaining in the location, the robot resumes the task. In the meantime, the adjacent robot monitoring this behaviour, appearing and disappearing in the vicinity, concludes that the task is initiated, hence it starts the task. This propagates through all robots, and they initiate the task one after another.

In all four proposed approaches, two shared problems have been identified: collision avoidance, and resource conflict. For collision avoidance, the artificial potential function is implemented. Also, to prevent the robots to enter the field as a result of the collision avoidance, an artificial potential force is applied if the distance to the field is less than a threshold.

To resolve the resource conflict, a robot has to decide to either give priority to another robot or to ignore the presence of others and takes the path. This decision has to be made only on the basis of the local information of the robot. For this, first the field of view of the robot is divided into three regions: Left Region (**LR**), Front Region (**FR**), and Right Region (**RR**). Then, using the following conventions, a robot can decide whether to take priority or not.

- **Left Convention**: If another robot is detected on the left region (LR), ignore and take the priority.
- **Right and Front Convention**: If a robot is detected in front (FR) or on

the right region (RR), wait (i.e. stop with zero velocity) and give priority to the detected robot.

However, sensors are subject to noise, and they cover only a limited range. This, sometimes, leads to robots remaining undetected in the regions of interest. This is common when the robots are very close to the shared location. To rectify this, a threshold distance is defined on the left region. With this, the detected robots will be considered on the left region if the distance becomes less than a certain threshold distance.

The conventions are only effective when the robots are approaching from the same direction. However, it is in-effective when the robots are approaching from opposite directions since both robots stop for the path to be cleared.

There are two reasons why congestion still occurs in this situation: (I) The defined conventions make the robots to wait for another robot to take initiatives to clear the path. (II) Robots behave homogeneously (i.e. robots behave the same upon encountering another robot in the aforementioned regions). In order to break the homogeneity in robots' behaviour, instead of waiting at the time of detection, the robots move reversely with random velocity until the triggered region is cleared (i.e. the robot is no longer in sight).

5.2 Region-based vs Self-organised, FIFO, LIFO

In this research, four cooperative area coverage algorithms are introduced for two different methods of interaction. In one, the group interacts via changes in the environment as a result of execution of the task (chapter 3), and in the other, via sensing and monitoring one another (chapter 4).

The considered applications (spraying and ploughing) had limitations which did not

allow for both approaches to be implemented. However, there are other applications in which both approaches can be applied. For example, in beacon distribution application, robots can either collect information by observing each others behaviour or by detecting the planted beacons in the environment. In this situation, which approach is more appropriate? In this section, all proposed approaches are compared with each other.

5.2.1 Execution Time Comparison

One of the main elements of comparison is execution time. That is the amount of time required to cover and process the entire field. This includes both the task allocation and execution.

In all four proposed approaches, robots have to identify their share of the task in real-time and during execution time. This process is repeated for each robot at the end of each turn in self-organising ploughing approach. Whereas in FIFO, LIFO and spraying (hereafter refer to as the region-based), it occurs only once and at the beginning of the task. This decreases the overall execution time for the region-based approach.

However, as the number of robots increases, the task allocation duration for the region-based, FIFO, and LIFO approach increases. This is due to the fact that task allocation in these three approaches is performed sequentially and the resulted delay accumulates and propagates over the system. Subsequently, all four approaches will have more or less the same execution time.

The results obtained from the simulation of both systems in the Stage simulation environment using ROS API suggest that for smaller team sizes (between 3 and 10), using the region-based approach, a field with 51 process locations (furrows or

tracks) can be covered and processed at shorter period of time (see Figure 5.1). However, with larger team sizes (between 11 and 20), this advantage goes away and both approaches perform equally.

Note that during data collection, the robots' initial positions were fixed around the field. It started when the first robot reached the first point of interest (first ploughing location for the self-organised ploughing, and first checkpoint for the region-based). The data collection stops when the last robot completes the current task and leaves the field.

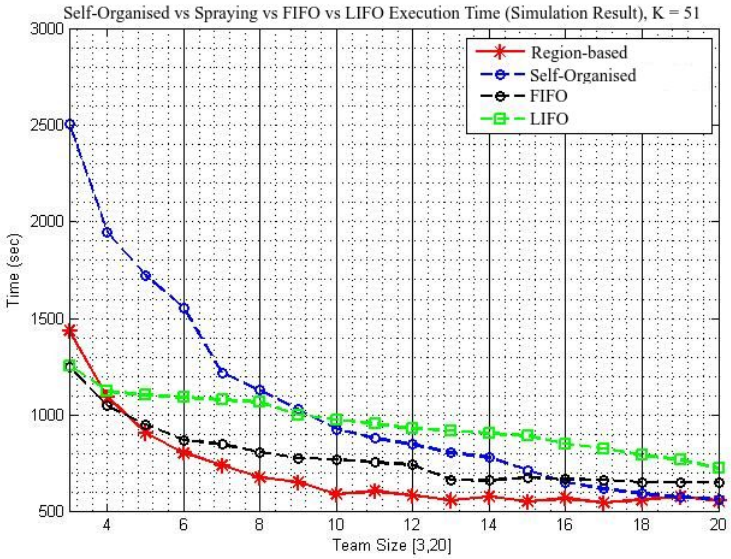


Figure 5.1: Comparison between the self-organised, FIFO, LIFO, and the region-based in terms of execution time for team sizes between 3 and 20 robots. Number of furrows in the field = 51. The Stage simulation results plotted using Matlab.

5.2.2 Scalability, Flexibility and the Required Coordination

Another element of comparison is scalability and flexibility. That is the level of tolerance when a single or a group of robots are added or removed during execution time. The level of scalability and flexibility of the system can be determined by the amount of adjustment after the change is made. The main area of impact when

a robot is added or removed is task allocation. In other words, when such change is made, the following question has to be answered: *which robot(s) will cover the unallocated portion of the task?*

In FIFO and LIFO, the task allocation occurs only once (at the first round of execution) with a fixed number of robots in the team. Any changes in team size at any point during execution (e.g. a robot joins the team, or a faulty robot breaks down and get removed from the team) cannot be tolerated and the field will never get processed. It could be said that both FIFO and LIFO are not flexible, and any changes to the team requires design time adjustments.

In the region-based approach, robots also perform task allocation only once at the beginning. However, unlike FIFO and LIFO in which robots in advance have team size as global knowledge, in the region-based the team size is unknown to the participating robots. This improves the flexibility of the system to a certain extent. However, if a robot is removed from the team after the task allocation is completed, part of the field will require extra and separate attention as no other robots will enter the field. It could be concluded that the region-based approach is flexible before task allocation, however, the system cannot recover itself if a robot is removed after task allocation is completed. On the other hand, adding a robot will not create major impact on the performance of the team, meaning that if a robot is added before task allocation, the robot may even be useful and replace a faulty robot, but after the task allocation, it has zero impact on the performance as it cannot participate in the task.

The self-organised approach has better scalability and flexibility compared to other approaches since the task allocation is occurring more than once for each robot (i.e. a robot's understanding of the task and the environment is refreshed at the end of

every round of execution) as task allocation and task execution are intertwined. In addition, like the region-based, the number of participating robots is not considered for task allocation. Therefore, at any point during task execution, while the robots are outside the field, if a new robot or a group of robots join the process, the overall execution time will decrease. This is the same with removing a robot or a group of robots during execution. In either case, no changes are required to be made on the behaviour of the robots and the system can fully recover if a robot is removed or added.

5.2.3 Resilience Toward Failure

In the real world, robots may fail due to various environmental causes, and it is important to consider how does a team can adapt to the new situation and complete the task. In the considered applications, for each proposed cooperative algorithm, the adaptability of the system is viewed based on the location where the failure has occurred.

The points of failure could be listed as follows:

1. **failure during task allocation** : robots failure to be part of the team and fail to identify its share of the task. This could be before task allocation or shortly after task allocation.

2. **failure during task execution** : robots fail to complete the allocated share of the task. At this point, the task is initiated.

The focus of this section is to view the flexibility of the proposed algorithms during and shortly after the task allocation. In any of the proposed approaches, failure during task execution has not been discussed. This is due to the limitations of both tasks (spraying and ploughing) thus robots are not allowed to evade any traffic while

in the field (*reminder:* while in the field, evasive manoeuvres of the robots causes the already processed part of the field to be run over. This will destroy the crop (in spraying) and the soil pattern (in ploughing)).

However, the effect of this failure is different. If any robot fails while performing the actual task of ploughing, the following robots will fail to resumes as the robots cannot manoeuvre and pass the faulty robot. In other cases, the other robots can treat the faulty one as a static obstacle and perform normal obstacle avoidance while switching the processing track. In the task of spraying, since the robots process independent sections of the field, if a robot fails during the execution, other robots can successfully complete their share of the task. Therefore, the failure tolerance is slightly higher.

In both the FIFO and LIFO approaches, the team size is part of the common knowledge (i.e. every robot in the team knows and relies on the number of participating robot). Using this information, robots in both approaches estimate where to plough next. Since in ploughing there is the sequential limitation (i.e. one furrow has to be followed by a ridge), robot will never be able to successfully create this pattern on the ground if the predefined team size is violated. This is because the execution initiation signal of a robot comes from the previous robot's execution result (i.e. each robot has to make sure that the previous robot either has completed the current ploughing or already started a new round of ploughing). Therefore, any failures in individuals execution (including sudden death of a robot, or addition of robots) will be propagated throughout the team, and the field will not be ploughed.

Unlike FIFO and LIFO, in the self-organised approach, the task allocation does not depend on the number of participating robots. In other words, when losing one or more robots at any point during task allocation others can resume the task and

obtain where to plough next.

In the region-based approach, the task allocation is dependent. In other words, if a robot fails during this stage, the task may never be initiated. However, once the task has begun, the failure of one robot during execution does not affect other robots.

5.3 Application Scope

In this thesis, two cooperative algorithms are suggested for the area coverage applications which rely only on local sensing. The considered field applications are spraying and ploughing. There are more field applications (in area coverage category) which can benefit from the proposed algorithms including seeding, harvesting, de-mining, beacon distribution, aggregation and etc. But which algorithm is suitable for these field of applications? that is the question that is going to be answered in this section.

5.3.1 Seeding

According to Fao in 1997, "the location of plants In a furrow system is not fixed but depends on the natural circumstances". In areas with heavy rainfall, the plants should stand on top of the ridge in order to prevent damage as a result of waterlogging (Figure 5.2(a)). If water is scarce, the plants may he put in the furrow itself, to benefit more from the limited water (Figure 5.2(b)). As salts tend to accumulate in the highest point, a crop on saline soils should be planted away from the top of the ridge. Usually it is planted in two rows at the sides (Figure 5.2(c)). For winter and early spring crops in colder areas, the seeds may be planted on the sunny side of the ridge (Figure 5.2(d)). In hotter areas, seeds may be planted on the shady side of the ridge, to protect them from the sun.

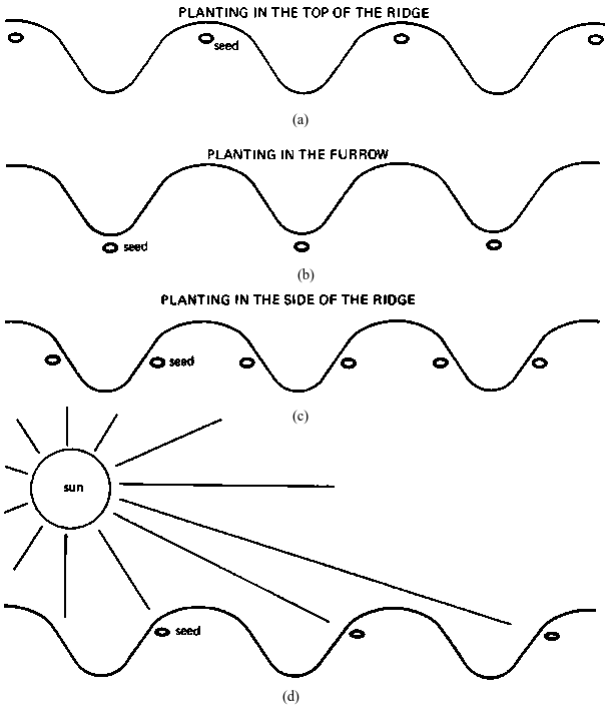


Figure 5.2: Seeding Methods (Fao, 1997): (a) ridge seeding. (b) furrow seeding. (c) side ridge seeding. (d) side ridge seeding sun movement

In some seeding approaches, it is possible to detect seeds after it is planted, and in others this feature does not exist. Depending on seed detecting feature, either of the self-organised, or the region-based approach can be applied. Note that in the self-organised approach, the robots, instead of searching for soil pattern changes, search for existence of seeds at particular pre-defined locations.

5.3.2 Harvesting

Harvesting is the process of gathering a ripe crop from the fields. This involves navigation of the harvesting unit throughout the field. The navigation pattern of the harvesting unit depends on the crop. Root crops including onions, potatoes, carrots, and beat roots are harvested in rows. In this approach, the harvester has to follow the rows and lift the crop out of the soil (Figure 5.3). Whereas when harvesting grains, the harvester unit can navigate in any direction.

Figure 5.3: Harvesting Carrot (Radu, 2014): the harvesting machine moves through the rows and dig out carrots

In this process for each group of crops, different harvesting tools are required which in return expects a different form of behaviour. Generally, harvesting machineries can be classified into three different categories: Human-Machine Cooperation in which human is involved in actual picking of the crop (Figure 5.4(a)), Machine-Machine Coordination in which there is at least one harvester in cooperation with a group of containers (Figure 5.4(b)), and Self-Containing Machine Harvesters in which there is only one group of machines involved in the process (Figure 5.4(c)).

(a) (b) (c)

Figure 5.4: Harvesting Machines: (a) human-machine cooperation harvesting (Haygrove, 2017). (b) machine-machine cooperation (Deere, 2014). (c) self-contained harvesters (Landwirt, 2017)

Heterogeneous Team for Harvesting A visit has been made to Noord Oost Polder in the Netherlands during carrot and beet roots harvesting period. Interviews were conducted with the farmers and the harvesting operation was observed. Based on our field investigation, when harvesting carrots or beet roots (to name a few), normally a harvester along with a group of container units are deployed into the field. The harvester takes the lead and navigates through the field. The containers, one after another, pose themselves in a way that they can receive the load from the harvester. Only when the container robot is in position the harvester resumes with the operation.

From a multi robotic perspective, this is a heterogeneous team since there are two kinds of robots in the team (i.e. they are different in software and hardware). Although the considered team in the proposed approaches (the region-based and the self-organised approach) is homogeneous (i.e. the participating robots have the same hardware and software), with slight modification the self-organised approach can be applied in here. As there is only one harvester and it processes the field sequentially, the region-based approach is less efficient and inappropriate.

The reason is the limitation in the participation of robots. Unlike spraying and ploughing in which robots can participate with no self-defined limitations, in harvesting robots can participate as much as the container capacity. In spraying and ploughing, robots can resume the task for the entire region or furrow, whereas their capacity may force them to leave the task after a short period of participation. In this situation, the more efficient approach is the one in which there are other robots to replace the full-capacity robot and fill the delay gap.This delay cannot be covered using the region-based approach, as it is demonstrated in Figure 5.5. Whereas, robots in self-organising approach can recover each other (see Figure 5.6).

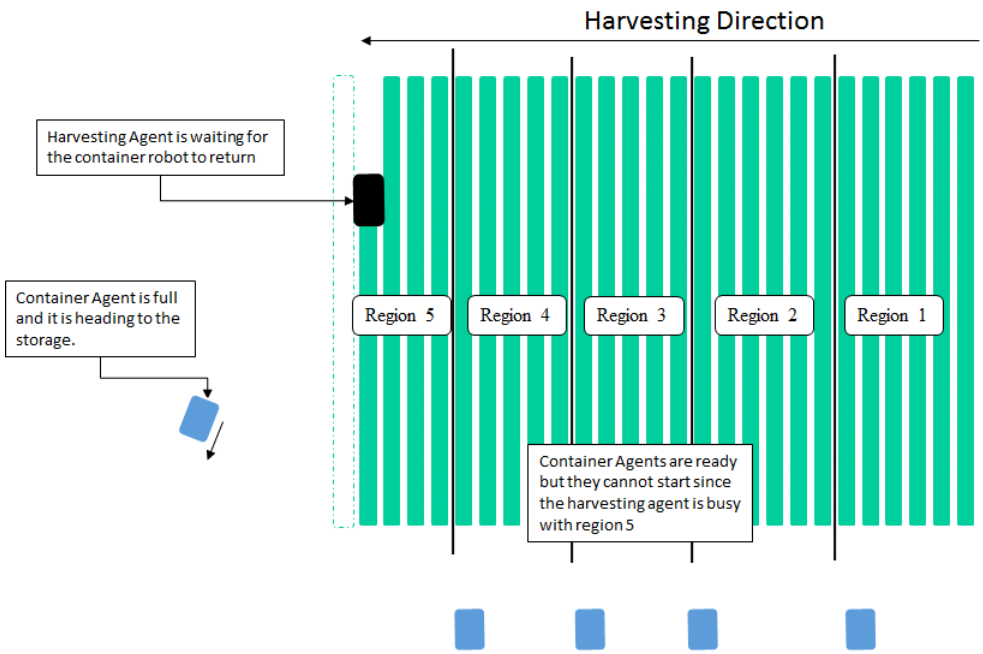


Figure 5.5: Single Harvester Region-based Approach: robots on other regions cannot contribute to the task when other regions are being processed

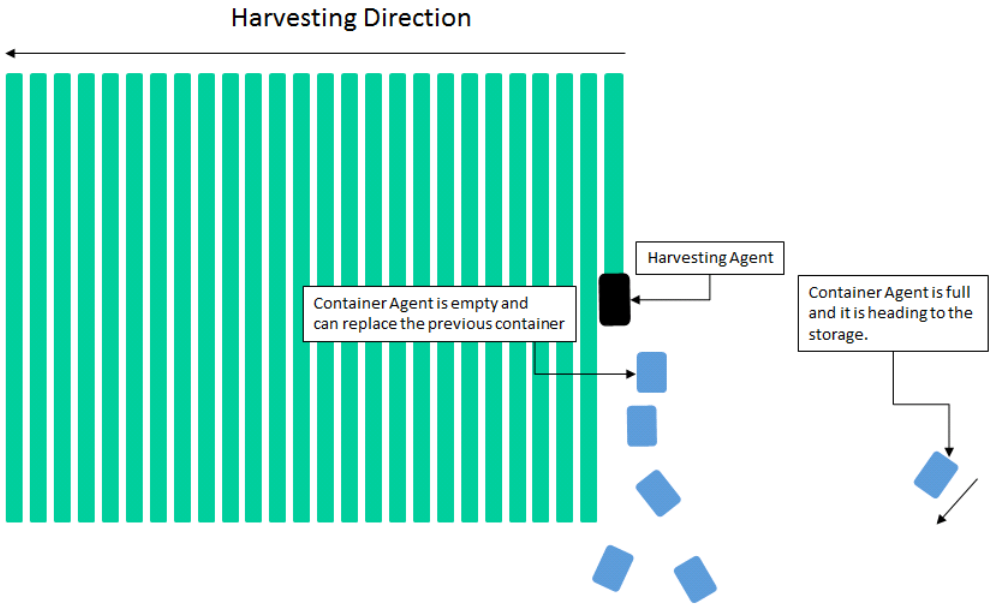


Figure 5.6: Single Harvester Self-organising Approach: task progress does not depend on the progress of another region or track. All robots can participate at any location in the field if the harvester is available.

Homogeneous Team for Harvesting When each harvester becomes capable to harvest and store the crop within itself, from the multi robotic perspective, it becomes a homogeneous team (since there is only one role in the team and hence the required software and hardware is the same). In this situation, all available approaches can be applied. Note that redundancy in the harvesting has no negative effect on the crop, unlike spraying.

5.3.3 Multi-Robotic De-mining and Mine Field Mapping

Landmines have killed more than one million people since 1975 (Baudoin et al., 1999). It is estimated that every year between 15000 and 25000 people are maimed or killed by landmines (Walsh and Walsh, 2003). Interesting to know that the cost of making one landmine is $3 - $10 whereas deactivating one cost between $300 - $1000. Demining operations are deadly operations. According to Baudoin et al. (1999) for every 5000 cleared mines, there is one deminer killed. In order to clear some 60,000,000 mines, we could count some 12,000 deminers killed. This logic attracted many human free operations including human-rat cooperation (Nanayakkara et al., 2008), and teleoperated robotics such as Uran-6 (Army-Technology, 2016) developed by the Russian army.

During the de-mining operation, the field is investigated for TNT signatures and depending on the approach either the location is pinpointed or the mine is disarmed. Since the unit has to deal with explosives, the failure is highly probable, hence multi robot system and redundancy in the approach increases the chance of success.

(a) (b)

Figure 5.7: human-free demining: (a) Rats are trained to detect mines (Sumocakewalk, 2015). (b) Uran-6, the latest Russian demining unit deployed in Syria which can tolerate upto 20 KG TNT explosion (More, 2016).

Since landmine deactivation is an open field operation, and redundancy is an advantage as the chances of losing an agent due to dealing with explosives is high, the use of the region-based algorithm is recommended. This is solely due to the fact that the region-based approach has higher flexibility and tolerance toward loss of an agent during task execution. In other proposed approaches including FIFO, LIFO and the self-organised method, the failure of an agent has impact on the task progress, thus they are not appropriate for this task.

5.3.4 Cooperative Beacon Distribution

Another important field operation that can benefit from the proposed approaches is the distribution of beacons in an open field. In this application, a group of robots has to distribute themselves (if they are the sensor) or they loaded sensors throughout the field. One example of this application is localisation for underwater robotics. Underwater robotic requires localisation system since GPS signals cannot be reached under the water. One method is to distribute GPS beacons on the surface of the water as GPS receiving points.

At the first glance, it seems that both approaches can be applied in this application. But with a slight modification: instead of detecting pattern on water (furrow-ridge), the approach should look for beacons. But there is one major drawback which prevents this approach to work. The movement on the surface of the water makes the self-organise approach to be less reliable due to movement on the surface of the water. However, the region-based approach provides more reliable result compare to the self-organised approach since does not require analysis of the changes in the environment at particular locations.

5.4 Conclusion and Future Directions of Research

In this research, we provided four cooperative algorithms with which a team of robots can cover an open field cooperatively with restrictions defined by two fundamentally different tasks, spraying, and ploughing. In the design of each cooperative algorithm we aimed to answer the following questions:

Task Allocation and Interaction : Which robot is processing which part of the field? and how does each robot is informed about the state of the task and the field?

Coordination and Interaction : How robots detects and avoid obstacles? and how robots reach the targeted location one by one without congestion? (i.e. In case two or more robots want to access the same coordinate at the same time, which one will get there first?)

Methods using explicit forms of communication for coordination and interaction was not an option. This is due to the fact that these extensively studied approaches suffer from losing the central organiser or the communicating signal, and short range of signal coverage which makes them inappropriate for large fields. Also, they all assume that initially robots are located within the communication range which becomes invalid if robots are distributed around the field executing other tasks prior to the execution time.

Instead, the focus was to answer these research questions relying only on robot local sensing (i.e. interaction through the environment and through agent detection). However, a single approach could not be applied for both considered tasks since different task have different characteristics. In ploughing, for example, as a result of execution, a particular pattern is created on the soil. This pattern exists in seeding

or spraying from the beginning, and if the ploughing interaction method is applied in spraying, it will cause destructive results (*reminder:* excessive chemical spraying on a same location). Therefore the logic behind team interaction could be in two ways: either robots detect previous robots trail at particular locations, or they monitor each other behaviour throughout the process.

As a result of ploughing, a 2D vision-based algorithm was developed to detect the created pattern on the soil at particular locations. And in ploughing, a 2D vision-based and laser-based algorithm were developed to detect robots behaviours throughout the process.

Artificial Potential Field (APF) method is applied with which robots utilise the head-mounted laser range finder to navigate through the field while avoiding obstacles. Using this method, robots also prevented to enter the field as the state of the task may be unknown to the robots. For this, massive artificial repulsion force is assigned to the border of the field which redirects the resultant sum force.

It also understood that relying solely on the APF is not sufficient for navigation as all robots aim to reach the same location at once. This is a known problem and it is referred to as Spatial Resource Conflict. To solve this issue, an algorithm, Random Reverse Velocity (RRV), was developed and infused with APF.

Combining these algorithms, four approaches was introduced: FIFO, LIFO, and the self-organised for ploughing and the region-based approach for spraying. In FIFO, LIFO and the self-organised approaches, the task allocation is carried out based on robots visiting ploughing locations and monitoring for a particular pattern (furrow-ridge) on the soil. The key difference between FIFO, LIFO and the self-organised approach is the knowledge of the number of participating robots. In FIFO and LIFO, the number of participating robots is fixed in other words it has no real-time

scalability characteristics. That makes the task allocation really fast, but at the same time makes the approaches fragile since it needs every single robot to perform at their best. Therefore, if the approach is applied in other applications in which the state of the robots can be guaranteed, FIFO and LIFO provide faster performances. The self-organised approach, on the other hand, is more reliable for tasks in which robots cannot be trusted as it provides more flexibility toward loss of an agent during task allocation.

In the region-based approach, the task allocation is carried out by claiming particular locations outside of the regions which are known to all robots. The robots visiting these locations see if the location is free. In this case, they proceed to occupy the location otherwise they move on to the next location. They continuously monitor the robot in the next location to determine when to start their task. Once the task is started, the robots resume the process independently. This makes the region-based approach more flexible towards loss of an agent during task execution in contrast with other three approaches in which failure in the middle of execution prevents the task to be completed. However, it is less flexible that the self-organised approach toward loss of an agent during task allocation process as task allocation is a dependent task in the region-based approach.

The proposed approaches are not solely designed for spraying and ploughing and they can be applied to other applications including seeding, harvesting, de-mining, line searching, aggregation, beacon distribution and any other open field operations which require a team.

Even though the simulation results in stage simulation environment with ROS is promising, there are certain factors which are missing when simulating. These improving factors are considered as the future direction of research and they are listed

below:

5.4.1 Recovering from Failure During Task Execution

System optimisations introduced in chapter 3 and 4 make both the self-organised and the region-based approach more resilient toward certain failure during task allocation. However, if a robot fails during task execution in the middle of the field, the task will never be completed successfully. Somehow other robots have to be informed about this situation and an appropriate action has to be taken. For applications other than ploughing, evading the faulty robot and resuming the task could be an option, but the outcome is undesirable (now you are left with a processed field with a faulty robot in the middle).

5.4.2 Hybrid Approaches

From the beginning of this research, the main aim was to obtain the cooperation without the use of direct communication. Despite the promising results, one cannot deny and ignore the impact of direct communication in the performance of the team. Although relying on direct communication for task allocation is still recommended to be avoided due to initial assumption mentioned in chapter 3, short range direct communication (e.g. via RGB LED signals, ultrasound messaging) can still be integrated for coordination and error correction purposes.

The followings are the foreseen areas in which short-range direct communication can improve:

Localisation Correction: robots can correct their positioning system in case of losing the GPS signal or whenever their confidence is dropped.

Region-based Task Allocation Optimisation: The region-based approach they

can reduce the region analysis time by broadcasting the region number corresponding to the occupying checkpoint.

Self-organising Knowledge Update: In the self-organised approach, each robot builds its own knowledge since they solely rely on their field investigation. With the help of short-range communication, robots which are spending more time in the field can broadcast their knowledge (e.g. the last location number or coordinate that was processed, or the field has completely processed no need for analysis) to whichever robot nearby. In this way, the field can be processed much quicker and more cost-effectively.

Recovering From Failure During Task Execution: More importantly, short range direct communication can be a quick solution for recovering from a failure during task execution, particularly in ploughing. Since in ploughing, robots are operating close to each other, robots which are trapped or stuck in the middle of the field can benefit from short range direct communication to ask for assistance. The message can be propagated throughout the field and appropriate strategies can take place. However, short-range direct communication cannot resolve this issue if happens in the region-based approach since robots are regions apart from each other. Unless the field is partitioned in a way that robots can be in communication range.

References

AAPlus (2017). Pest control courses.

Agassounon, W. and Martinoli, A. (2002). Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. pages 1090–1097.

Agriavis (2015). Mouldboards. `http://www.agriavis.com/famille-115-charrues-1.html`.

Ahmadi, M. and Stone, P. (2006). A multi-robot system for continuous area sweeping task. *In Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1724–1729.

Aiswarya, M., Farza, P., Kumar, S. S., Sneha, S., and Varughese, T. A. (2016). Automatic weed detection system and smart herbicide sprayer robot for corn fields. *IJRCCT*, 5(2):055–058.

Alexandratos, N. and Bruinsma, J. (2012). World agriculture towards 2030/2050: The 2012 revision. *Food and Agriculture Organization of the United Nations*, pages 1–154.

Anil, H., Nikhil, K., Charitra, V., and Gurusharan, B. (2015). Revolutionizing farming using swarm robotics, modelling and simulation. *6th International Conference on Intelligent Systems*, 19(3):141–147.

Arbanas, B., Ivanovic, A., Car, M., Haus, T., Orsag, M., Petrovic, T., and Bogdan, S. (2016). Aerial-ground robotic system for autonomous delivery tasks. pages 5463–5468.

Arguenon, V., Bergues-Lagarde, A., Rosenberger, C., Bro, P., and Smari, W. (2006). Multi-agent based prototyping of agriculture robots. pages 282–288.

Arkin, R. (1998). *Behaviour-based Robotic*. The MIT Press, 1 edition.

Army-Technology (2016). Uran-6 mine-clearing robot, russia.

Arslan, G., Marden, J., and Shamma, J. (2007). Autonomous vehicle-target assignment: A game-theoretical formulation. *Journal of Dynamic Systems, Measurement, and Control*, 129(5):584–596.

Asama, H., Matsumoto, A., and Ishida, Y. (1989). Design of an autonomous and distributed robot system: Actress. *IROS*, 89:283–290.

Astrand, B. and Baerveldt, A. (2002). An agricultural mobile robot with vision-based perception for mechanical weed control. *Autonomous Robots*, 13(1):21–35.

Bakker, T. (2009). *An Autonomous Robot for Weed Control*.

Balta, H., Bedkowski, J., Govindaraj, S., Majek, K., Musialik, P., Serrano, D., Alexis, K., Siegwart, R., and Cubber, G. (2015). Integrated data management for a fleet of searchandrescue robots. *Journal of Field Robotics*.

Banfi, J., Li, A. Q., Basilico, N., Rekleitis, I., and Amigoni, F. (2016). Asynchronous multirobot exploration under recurrent connectivity constraints. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5491–5498.

Barca, J. C. and Sekercioglu, Y. A. (2013). Swarm robotics reviewed. *Robotica*, 13:345–359.

Barrientos, A. G., Lopez, J. L., Espinoza, E. S., Hoyo, J., and Palomo, G. V. (2016). Object transportation using a cooperative mobile multi-robot system. *IEEE Latin America Transactions*, 14(3):1184–1191.

Batalin, M. A. and Sukhatme, G. S. (2002). Spreading out: A local approach to multi-robot coverage. pages 373–382.

Baudoin, Y., Acheroy, M., Piette, M., and Salmon, J. (1999). Humanitarian demining and robotics. *Mine Action Information Journal issue on Machine Assisted Demining*, 3(2).

Beck, Z., Teacy, L., Rogers, A., and Jennings, N. (2016). Online planning for collaborative search and rescue by heterogeneous robot teams. *In Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems - International Foundation for Autonomous Agents and Multiagent Systems*, pages 1024–1033.

Bell, T. (2000). Automatic tractor guidance using carrier-phase differential gps. *Computers and electronics in agriculture*, 25(1):53–66.

Billingsley, J. and Schoenfisch, M. (1997). The successful development of a vision guidance system for agriculture. *Computers and electronics in agriculture*, 16(2):147–163.

Blackmore, S. (2012). Robotic agriculture: Designing systems for the farm of tomorrow. `https://www.innovateuk.org/c/document_library/get_file?groupId=2828839&folderId=7196297&title=3+Simon+Blackmore+Robotic_Agriculture.pdf`.

Bloss, R. (2014). Robot innovation brings to agriculture efficiency, safety, labor

savings and accuracy by plowing, milking, harvesting, crop tending/picking and monitoring. *Industrial Robot: An International Journal*, 41(6):493–499.

Borenstein, J., Everett, H. R., Feng, L., and Wehe, D. (1997). Mobile robot positioning-sensors and techniques.

Borenstein, J. and Feng, L. (1996). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on robotics and automation*, 12(6):869–880.

Botelho, S. C. and Alami, R. (1999). M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. 2:1234–1239.

Caloud, P., Choi, W., Latombe, J. C., Le Pape, C., and Yim, M. (1990). Indoor automation with many mobile robots. *Intelligent Robots and Systems' 90.'Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on*, pages 67–72.

Castello, E., Yamamoto, T., Dalla Libera, F., Liu, W., Winfield, A., Nakamura, Y., and Ishiguro, H. (2016). Adaptive foraging for simulated and real robotic swarms: the dynamical response threshold approach. *Swarm Intelligence*, 10(1):1–31.

Chandrasekaran, B., Annadurai, K., and Somasundaram, E. (2010). *A Textbook of Agronomy*. New Age International, 1 edition.

Chawla, K. and Robins, G. (2011). An rfid-based object localisation framework. *International journal of radio frequency identification technology and applications*, 3(1-2):2–30.

Choi, J. S. S., Lee, H., Elmasri, R., and Engels, D. W. (2009). Localization systems using passive uhf rfid. pages 1727–1732.

Choi, K. H., Han, S. K., Park, K. H., Kim, K. S., and Kim, S. (2015). Vision based guidance line extraction for autonomous weed control robot in paddy field. *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 831–836.

Choset, H. (2001). Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1-4):113–126.

Chung, W. and Lau, E. (2007). Enhanced rssi-based real-time user location tracking system for indoor and outdoor environments. pages 1213–1218.

Clarke, L. (1997). Agricultural mechanization strategy formulation, concepts and methodology and the roles of the private sector and the government. *AGST, Food and Agriculture Organisation (FAO)*, 7:143–147.

Connolly, C. and Grupen, R. (1993). The applications of harmonic functions to robotics. *Journal of Robotic Systems*, 10(7):931–946.

Cottefoglie, F., Farinelli, A., Iocchi, L., and Nardi, D. (2004). Dynamic token generation for constrained tasks in a multi-robot system. *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, 1:911–917 vol.1.

Couceiro, M. (2015). An overview of swarm robotics for search and rescue applications. *Handbook of Research on Design, Control, and Modeling of Swarm Robotics*, page 345.

Dai, X., Jiang, L., and Zhao, Y. (2016). Cooperative exploration based on supervisory control of multi-robot systems. *Applied Intelligence*, pages 1–12.

Deere, J. (2014). John deere announces certified pre-owned equipment program. `https://investor.deere.com/our-company/`

investors-relations/news-releases/news-releases/2014/John-Deere-Announces-Certified-Pre-Owned-Equipment-Program/default.aspx.

Dias, M. B., Zlot, R., Kalra, N., and Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270.

Drenjanac, D. and Tomic, S. (2013). Middleware challenges in robotic fleets for precision agriculture. *In Recent Advances in Robotics and Mechateronics*, pages 23–34.

Drenjanac, D., Tomic, S., Klausner, L., and Kuhn, E. (2014). Harnessing coherence of area decomposition and semantic shared spaces for task allocation. *In Information Processing in Agriculture*, pages 23–34.

Dudek, G., Jenkin, M. R., Milios, E., and Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397.

English, A., Ball, D., Ross, P., Upcroft, B., Wyeth, G., and Corke, P. (2013). Low cost localisation for agricultural robotics. pages 1–8.

Estate, W. P. (2015). Three tractors ploughing together. https://www.youtube.com/watch?v=IcWuVXUvj9w.

Eurostat (2015). Agricultural labour input. http://ec.europa.eu/eurostat/statistics-explained/index.php/Archive:Agricultural_labour_input.

Faivre, S., Anderson, N., and Stelford, M. (2008). Agricultural automation system with field robot. US Patent App. 11/498,392.

Fao (1997). Chapter 3. furrow irrigation. http://www.fao.org/docrep/s8684E/s8684e04.htm.

Fao (2002). *World agriculture: towards 2015/2030 Summary report*. FOOD AND AGRICULTURE ORGANIZATION OF THE UNITED NATIONS, 1 edition.

Fao (2009). Global agriculture towards 2050. *High Level Expert Forum*, pages 1–4.

Farinelli, A., Iocchi, L., and Nardi, D. (2004). Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2015–2028.

Fazeli, P., Davoodi, A., and Pasquier, A. (2010). Fault-tolerant multi-robot area coverage with limited visibility. *Proceedings of the ICRA 2010 Workshop on Search and Pursuit/Evasion in the Physical World: Efficiency, Scalability, and Guarantees*.

Fukuda, T. and Kawauchi, Y. (1990). Cellular robotic system (cebot) as one of the realization of self-organizing intelligent universal manipulator. *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 662–667.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Marín-Jiménez, M. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292.

Gerkey, B. and Mataric, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954.

Gerkey, B., Vaughan, R., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. *In Proceedings of the 11th international conference on advanced robotics*, 1:317–323.

Gerkey, B. P. and Mataric, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE transactions on robotics and automation*, 18(5):758–768.

Golait, R. (2007). Current issues in agriculture credit in india: An assessment. *Reserve Bank of India Occasional Papers*, 28:1–22.

Graca, R. A., Xiao, D., and Cheng, S. (2016). Multi-arm robotic painting process synchronization. US Patent 9,227,322.

Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes sociaux*, 6(1):41–80.

Graves, A., Morris, J., Deeks, L., Rickson, R., Kibblewhite, M., Harris, J., Farewell, T., and Truckle, I. (2015). The total costs of soil degradation in england and wales. *Ecological Economics*, 119:399–413.

Grift, T., Zhang, Q., Kondo, N., and Ting, K. (2008). A review of automation and robotics for the bio-industry. *Journal of Biomechatronics Engineering*, 1:37–54.

Guardian (2010). Guardian project. `https://bit.ly/2Qp2zdL`.

Gunn, T. and Anderson, J. (2015). Dynamic heterogeneous team formation for robotic urban search and rescue. *Journal of Computer and System Sciences*, 81(3):553–567.

Habibi, G., Xie, W., Jellins, M., and McLurkin, J. (2016). Distributed path planning for collective transport using homogeneous multi-robot systems. *In Distributed Autonomous Robotic Systems: Springer Japan*, pages 151–164.

Hague, T. and Tillett, N. (2001). A bandpass filter-based approach to crop row location and tracking. *Mechatronics*, 11(1):1–12.

Hansen, K. D., Garcia-Ruiz, F., Kazmi, W., Bisgaard, M., la Cour-Harbo, A., Rasmussen, J., and Andersen, H. J. (2013). An autonomous robotic system for mapping weeds in fields. *IFAC Proceedings Volumes*, 46(10):217–224.

Haygrove (2017). Pic-king 5 series. `http://www.haygrove.co.za/harvesting-machinery/harvest-rigs/pic-king-5-series/`.

Hewitt, C., Bishop, P., and Steiger, R. (1973). A universal modular actor formalism for artificial intelligence. *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245.

Hokuyo (2009). hokuyo-urg-04lx-ug01 data sheet. `http://www.robotshop.com/media/files/pdf/hokuyo-urg-04lx-ug01-specifications.pdf`.

Horna, R., Domialb, H., Slowihka-Jurkiewiczb, A., and Van-Ouwekerk, C. (1995). Soild compaction processes and their effects on the structure of arable soils and the environment. *Soil and Tillage Research*, pages 23–36.

Houmy, K., Clarke, L. J., Ashburner, J. E., and Kienzle, J. (2013). *Agricultural Mechanisation in Sub-Saharan Africa Guidlines for preparing a strategy.* Food and Agriculture Organisation.

Howard, A., Mataric, M. J., and Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. pages 299–308.

Huang, X., Janaswamy, R., and Ganz, A. (2006). Scout: outdoor localization using active rfid technology. pages 1–10.

Imou, K., Tani, S., and Yokoyama, S. (2009). Localization of agricultural vehicle using landmarks based on omni-directional vision.

Iocchi, L., Nardi, D., and Salerno, M. (2000). Reactivity and deliberation: a survey on multi-robot systems. *In Balancing reactivity and social deliberation in multi-agent systems*, 4:9–32.

ISwarm (2006). Guardian project. `https://www.shu.ac.uk/research/specialisms/materials-and-engineering-research-institute/what-we-do/projects/automation-and-robotics/iswarm-intelligent-small-world-autonomous-robots-for-micro-manipulation`.

Jager, M. and Nebel, B. (2001). Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 3:1213–1219.

Janani, A. (2015a). Ros-stage mrs area coverage, implicit form of interaction (lifo). `https://bit.ly/2OoUdFp`.

Janani, A. (2015b). Ros-stage mrs ploughing (with reversible moldboard) team of 8 robots (laser and camera) appraoch 1. `https://bit.ly/2y7Gzwa`.

Janani, A. (2015c). Ros-stage mrs spraying team of 10 robots (laser and camera). `https://bit.ly/2DKVtyI`.

Janani, A. (2016). Self-organising ploughing. `https://bit.ly/2OoRZWz`.

Janani, A. (2017). Artificial potential field ros stage simulation environment. `https://bit.ly/2OruUTh`.

Janani, A. (2018a). Apf on pioneer 3at and hokuyo lidar. `https://bit.ly/2xPuRXX`.

Janani, A. (2018b). Apf on pioneer 3at and hokuyo lidar(2). https://bit.ly/2OrvS1R.

Janani, A. (2018c). congestion avoidance: flee-opposite algorithm (with random reverse velocity). https://bit.ly/2RcwBm2.

Jeon, S., Jang, M., Lee, D., Cho, Y., Kim, J., and Lee, J. (2016). Multiple robots task allocation for cleaning and delivery. pages 195–214.

Jones, C. and Mataric, M. J. (2005). Behavior-based coordination in multi-robot systems. *Autonomous Mobile Robots: Sensing, Control, Decision-Making, and Applications*, pages 1–40.

Jose, K. and Pratihar, D. K. (2016). Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods. *Robotics and Autonomous Systems*, 80:34–42.

Jung, D. and Zelinsky, A. (2000). Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots*, 8(3):269–292.

K-Team (2017). Khepera iii. https://www.k-team.com/mobile-robotics-products/old-products/khepera-iii.

Karla, N. and Martinoli, A. (2006). A comparative study of market-based and threshold-based task allocation. *8th Symposium on Distributed Autonomous Robotic Systems*, pages 91–101.

Kato, S., Nishiyama, S., and Takeno, J. (1992). Coordinating mobile robots by applying traffic rules. *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, 3:1535–1541.

Khan, M. H., Li, S., Wang, Q., and Shao, Z. (2016). Distributed multirobot formation and tracking control in cluttered environments. *ACM Trans. Auton. Adapt. Syst.*, 11(2):1–22.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. of Robotics Research*, 5.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., and Matsubara, H. (1997). Robocup: A challenge problem for ai. *AI magazine*, 18(1):73.

Koch, P., May, S., Schmidpeter, M., Kühn, M., Pfitzner, C., Merkl, C., Koch, R., Fees, M., Martin, J., and Nüchter, A. (2015). Multi-robot localization and mapping based on signed distance functions. pages 77–82.

Koditschek, D. and Rimon, E. (1990). Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4):412–442.

Kong, C. S., Peng, N., and Rekleitis, I. (2006). Distributed coverage with multi-robot system. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2423–2429.

Krieger, M. J. B., Billeter, J., and Keller, L. (2000). Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406:992–995.

Landwirt (2017). Grimme kartoffelroder se 260. `https://www.landwirt.com/berichtdiashow/Agritechnica_2013_News_Erntetechnik,13,Grimme-Kartoffelroder-SE-260.html`.

Lee, J., Nam, Y., Hong, S., and Cho, W. (2012). New potential functions with random force algorithms using potential field method. *Journal of Intelligent and Robotic Systems*, 66(3):303–319.

Li, G., Yamashita, A., Asama, H., and Tamura, Y. (2012). An efficient improved artificial potential field based regression search method for robot path planning. *2012 IEEE International Conference on Mechatronics and Automation*, pages 1227–1232.

Li, L., Martinoli, A., and Y.S., A.-M. (2002). Emergent specialization in swarm systems. *Third International Conference Manchester, UK, August 12-14, 2002 Proceedings*, pages 261–266.

Li, N., Remeikas, C., Xu, Y., Jayasuriya, S., and Ehsani, R. (2015). Task assignment and trajectory planning algorithm for a class of cooperative agricultural robots. *Journal of Dynamic Systems, Measurement and Control*, 137(5):1–9.

Li, Q., Wang, L., Chen, B., and Zhou, Z. (2011). An improved artificial potential field method for solving local minimum problem. *Proceedings of 2011 2nd International Conference on Intelligent Control and Information Processing*, pages 420–424.

Liekna, A. and Grundspenkins, J. (2014). Towards practical application of swarm robotics: overview of swarm tasks. *In Proceedings of the 13th International Conference Engineering for Rural Development*, pages 271–277.

Liu, J. and Wu, J. (2001). *Multi-Agent Robotic Systems*. CRC Press, 1 edition.

Lopez-Gonzalez, A., Ferreira, E., Hernandez-Martinez, E., Flores-Godoy, J., Fernandez-Anaya, G., and Paniagua-Contro, P. (2016). Multi-robot formation control using distance and orientation. *Advanced Robotics*, pages 1–13.

Luke, S., Hohn, C., Farris, J., Jackson, G., and Hendler, J. (1998). Co-evolving soccer softbot team coordination with genetic programming. *Third International Conference Manchester, UK, August 12-14, 2002 Proceedings*, pages 398–411.

Lumelsky, V. and Harinarayan, K. (1997). Decentralized motion planning for multiple mobile robots: The cocktail party model. *Journal of Autonomous Robots*, 4:121–135.

Luna, R. and Bekris, K. E. (2011). Efficient and complete centralized multi-robot path planning. *In Proceedings of IROS'11, pages 3268-3275, San Francisco, CA, USA*.

Marchant, J. (1996). Tracking of row structure in three crops using image analysis. *Computers and Electronics in Agriculture*, 15(2):161–179.

Marchant, J. and Brivot, A. (1995). Real-time tracking of plant rows using a hough transform. *Real-Time Imaging*, 1(5):363–371.

Marcolino, L. and Chaimowicz, L. (2009a). Traffic control for a swarm of robots: Avoiding target congestions. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1955–1966.

Marcolino, L. S. and Chaimowicz, L. (2009b). Traffic control for a swarm of robots: Avoiding group conflicts. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1949–1954.

Marden, S. and Whitty, M. (2014). Gps-free localisation and navigation of an unmanned ground vehicle for yield forecasting in a vineyard.

Marjovi, A., Nunes, J. G., Marques, L., and de Almeida, A. (2010). Multi-robot fire searching in unknown environment. pages 341–351.

Mataric, M. J., Nilsson, M., and Simsarin, K. T. (1995). Cooperative multi-robot box-pushing. 3:556–561.

Mendoza, J., Biswas, J., Cooksey, P., Wang, R., Klee, S., Zhu, D., and Veloso, M. (2016). Selectively reactive coordination for a team of robot soccer champions. *Proceedings of AAAI-16 (2016, to appear)*.

Ming, L., Imou, K., Wakabayashi, K., and Yokoyama, S. (2009). Review of research on agricultural vehicle autonomous guidance. *International Journal of Agric Biol Eng*, 2:1–16.

More, R. Q. (2016). Russia's mine-clearing uran-6 robots to help get rid of hidden explosives in palmyra (videos).

Mousazadeh, H. (2013). A technical review on navigation systems of agricultural autonomous off-road vehicles. *Journal of Terramechanics*, 50:211–232.

Murphy, R. R. (2000). Marsupial and shape-shifting robots for urban search and rescue. *IEEE Intelligent Systems and their Applications*, 15(2):14–19.

Nanayakkara, T., Dissanayake, T., Mahipala, P., and Sanjaya, K. G. (2008). A human-animal-robot cooperative system for anti-personal mine detection.

Noguchi, N. and Barawid, O. (2011). Robot farming system using multiple robot tractors in japan agriculture. *Preprints of the 18th IFAC World Congress Milano*, 18(1):633–637.

Noguchi, N., Will, J., Reid, J., and Zhang, Q. (2004). Development of a master–slave robot system for farm operations. *Computers and Electronics in agriculture*, 44(1):1–19.

Noreils, F. R. (1993). Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79–98.

Okamoto, M. and Akella, M. (2016). Avoiding the local-minimum problem in multi-agent systems with limited sensing and communication. *International Journal of Systems Science*, 47(8):1943–1952.

Oksanen, T., Linja, M., and Visala, A. (2005). Low-cost positioning system for agricultural vehicles. pages 297–302.

Park, M., Jeon, J., and Lee, M. (2001). Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. *Industrial Electronics, 2001. Proceedings. ISIE 2001. IEEE International Symposium on*, 3:1530–1535.

Park, M. and Lee, M. (2003). Artificial potential field based path planning for mobile robots using a virtual obstacle concept. *Proceedings of 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 735–740.

Parker, C. A. C., Zhang, H., and Kube, C. R. (2003). Blind bulldozing: multiple robot nest construction. *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 2:2010–2015.

Parker, L. (1993). Designing control laws for cooperative agent teams. *Proceedings of the IEEE International Conference on Robotics and Automation, Hidden Valley, Pennsylvania*.

Parker, L. (1994a). Heterogeneous multi-robot cooperation (no. ai-tr-1465). *Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab*.

Parker, L. E. (1994b). Alliance: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots. *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, 2:776–783.

Parker, L. E. (1999). Cooperative robotics for multi-target observation. *Intelligent Automation and Soft Computing*, 5(1):5–19.

Penders, J. (1999). The practical art of moving physical objects. *Doctoral thesis*.

Penders, J., Alboul, L., Witkowski, U., Naghsh, A., Saez-Pons, J., Herbrechtsmeier, S., and El-Habbal, M. (2011). A robot swarm assisting a human fire-fighter. *Advanced Robotics*, 25(1-2):93–117.

Pierson, A. and Schwager, M. (2016). Adaptive inter-robot trust for robust multi-robot sensor coverage. *In Robotics Research: Springer International Publishing*, pages 167–183.

Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., and Birattari, M. (2011). Task partitioning in swarms of robots: An adaptive method for strategy selection. *Swarm Intelligence*, 3(3-4):283–304.

Prorok, A., Hsieh, M., and Kumar, V. (2016). Adaptive distribution of a swarm of heterogeneous robots. *Acta Polytechnica*, 56(1):67–75.

Prez, F. J. L. (2016). *Autonomous Navigation in an Indoor environment*.

Q. Bonnard, S. Lemaignan, G. Z. A. M. S. C. N. L. and Dillenbourg, P. (2013). Chilitags2: Robust fiducial markers for augmented reality and robotics. *CHILI, EPFL, Switzerland*.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. (2009). Ros: an open-source robot operating system. *In ICRA workshop on open source software*, 3(3.2):5.

Radu, M. (2014). Automated carrot harvester machines are mesmerizing to watch in action. `https://s1.cdn.autoevolution.com/images/news/`

automated-carrot-harvester-machines-are-mesmerizing-to-watch-in-action-videc
1.jpg.

Ramaithitima, R., Whitzer, M., Bhattacharya, S., and Kumar, V. (2016). Automated creation of topological maps in unknown environments using a swarm of resource-constrained robots. *IEEE Robotics and Automation Letters*, 1(2):746–753.

Ranjbar-Sahraei, B., Weiss, G., and Nakisaee, A. (2012). A multi-robot coverage approach based on stigmergic communication. *German Conference on Multiagent System Technologies*, pages 126–138.

Rao, A., F., A. B., Lindgren, A., and Ziviani, A. (2016). Team communication strategy for collaborative exploration by autonomous vehicles. *2016 IEEE International Conference on Communications (ICC)*, pages 1–6.

REINS (2011). Reins project. https://www.shu.ac.uk/research/specialisms/materials-and-engineering-research-institute/what-we-do/projects/automation-and-robotics/the-reins-project.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH computer graphics*, 21(4):25–34.

RHEA (2014). Rhea project, robot fleets for highly effective agriculture and forestry management. http://www.rhea-project.eu/.

Robin, C. and Lacroix, S. (2016). Multi-robot target detection and tracking: taxonomy and survey. *Autonomous Robots*, 40(4):729–760.

Roldan, J. J., Garcia-Aunon, P., Garzon, M., de Leon, J., del Cerro, J., and Barri-

entos, A. (2016). Heterogeneous multi-robot system for mapping environmental variables of greenhouses. *Sensors*, 16(7):1018.

Ruiz-Garcia, L. and Lunadei, L. (2011). The role of rfid in agriculture: Applications, limitations and challenges. *Computers and Electronics in Agriculture*, 79(1):42–50.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2 edition.

Sabanci, K. and Aydin, C. (2017). Smart robotic weed control system for sugar beet. *Journal of Agricultural Science and Technology*, 19(1):73–83.

Saez-Pons, J., Alboul, L., Penders, J., and Nomdedeu, L. (2010). Multi-robot team formation control in the guardians project. *Industrial Robot: An International Journal*, 37(4):372–383.

Safadi, H. (2007). Local path planning using virtual potential field. `http://www.cs.mcgill.ca/~hsafad/robotics/`.

Sahin, E. (2004). Swarm robotics: From sources of inspiration to domains of application. pages 10–20.

Sahin, E., Girgin, S., Bayindir, L., and Turgut, A. (2008). Swarm robotics. *In Natuarl Computing Series, chapter Swarm Intelligence*, pages 87–100.

Santana, P., Barata, J., Cruz, H., Mestre, A., Lisboa, J., and Flores, L. (2005). A multi-robot system for landmine detection. 1:8–pp.

Saska, M., Vonásek, V., and Přeučil, L. (2013). Trajectory planning and control for airport snow sweeping by autonomous formations of ploughs. *Journal of Intelligent & Robotic Systems*, 72(2):239–261.

Sen, A., Sahoo, S., and Kothari, M. (2016). A cooperative target-centric formation with bounded acceleration. *IFAC-PapersOnLine*, 49(1):425–430.

Senanayake, M., Senthooran, I., Barca, J., Chung, H., Kamruzzaman, J., and Murshed, M. (2016). Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems*, 75:422–434.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Shearer, S. A. and Pitla, S. K. (2013). Field crop production automation. *Agricultural Automation*, pages 99–124.

Shearer, S. A., Pitla, S. K., and Luck, J. (2010). Trends in the automation of agricultural field machineries. *Biosystems and Agricultural Engineering, University of Kentucky, Lexington, USA*, pages 1–21.

Simmons, R., Apfelbaum, D., Burgard, W., Fox, D., Moors, M., Thrun, S., and Younes, H. (2000). Coordination for multi-robot exploration and mapping. pages 852–858.

Simmons, R., Singh, S., Hershberger, D., Ramos, J., and Smith, T. (2001). First results in the coordination of heterogeneous robots for large-scale assembly. *Experimental Robotics VII*, pages 323–332.

Slaughter, D., Chen, P., and Curley, R. (1996). Vision guided precision cultivation. *Precision Agriculture*, 1(2):199–217.

Smith, R. (1980). The contract net protocol: high-level communication and control in distributed problem solver. *IEEE Trans. Computers*, C-29(12):1104–1113.

Sojka, R., Bjorneberg, D., and Entry, J. (2002). Irrigation: A historical perspective. *Marcel Dekker, Inc.*

Solovey, K., Salzman, O., and Halperin, D. (2015). Finding a needle in an exponential haystack: Discrete rrt for exploration of implicit roadmaps in multi-robot motion planning. pages 591–607.

Stoll, A. and Kutzbach, H. D. (2000). Guidance of a forage harvester with gps. *Precision Agriculture*, 2(3):281–291.

Sujaritha, M., Lakshminarasimhan, M., Fernandez, C. J., and Chandran, M. (2016). Greenbot: A solar autonomous robot to uproot weeds in a grape field. *International Journal of Computer Science and Engineering*, 4(2):1351–1358.

Sukkarieh, S. (2012). Automated agriculture. `http://sydney.edu.au/news/84.html?newsstoryid=10737`.

Sumocakewalk (2015). Cambodia imports african rats for demining work.

Sungjun, A., Youdan, K., and Jaemyung, A. (2015). Area allocation algorithm for multiple uavs area coverage based on clustering and graph. *1st IFAC Workshop on Advanced Control and Navigation for Autonomous Aerospace Vehicles ACNAAV'15*, 45(9):204–209.

Sgaard, H. and Olsen, H. (2003). Determination of crop rows by image analysis without segmentation. *Precision Agriculture*, 38(2):141–158.

Tang, F. and Parker, L. E. (2005). Asymtre: Automated synthesis of multi-robot task solutions through software reconfiguration. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1501–1508.

Tang, J., Geng, N., Zhang, Z., and Zhu, Z. (2013). A vision-based method of wheat row detection for agricultural robot. *International Journal of Digital Content Technology and its Applications*, 7(5):129.

Tarannum, N., Rhaman, K. M., Ahmed Khan, S., and Shakil, R. (2015). A brief overview and systematic approach for using agricultural robot in developing countries. *Journal of Modern Science and Technology*, 3:88–101.

tarmakbir (2017). Last day for grant support in irrigation systems 14 feburary (sulama sistemlerinde hibe destei in son gn 14 ubat). US Patent App. 11/498,392.

Thrun, S. and Liu, Y. (2005). Multi-robot slam with sparse extended information filers. pages 254–266.

Thuilot, B., Cariou, C., Cordesses, L., and Martinet, P. (2001). Automatic guidance of a farm tractor along curved paths, using a unique cp-dgps. 2:674–679.

Tian, Z., Junfang, X., Gang, W., and Jianbo, Z. (2014). Automatic navigation path detection method for tillage machines working on high crop stubble fields based on machine vision. *International Journal of Agricultural and Biological Engineering*, 7(4):29.

Tillett, N., Hague, T., and Miles, S. (2002). Inter-row vision guidance for mechanical weed control in sugar beet. *Computers and Electronics in Agriculture*, 33(3):163–177.

Todt, E., Rausch, G., and Suarez, R. (2000). Analysis and classification of multiple robot coordination methods. *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 4:3158–3163.

Tsai, C., Chan, C., and Tai, F. (2015). Cooperative localization using fuzzy decentralized extended information filtering for homogenous omnidirectional mobile multi-robot system. *New Trends on System Science and Engineering: Proceedings of ICSSE 2015*, 276:343.

Uny Cao, Y., Fukunaga, A. S., and Kahng, A. B. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:1–23.

Vig, L. and Adams, J. A. (2006). Multi-robot coalition formation. *IEEE transactions on robotics*, 22(4):637–649.

Vlassis, N. (2003). *A Concise Introduction to Multiagent Systems and Distributed AI*. Universiteit van Amsterdam, 1 edition.

Walsh, N. E. and Walsh, W. S. (2003). Rehabilitation of landmine victims: the ultimate challenge. *Bulletin of the World Health Organization*, 81(9):665–670.

Wang, J. and Olson, E. (2016). Apriltag 2: Efficient and robust fiducial detection. pages 4193–4198.

Weber, J., Kaufmann, C., Hache, C., and Schmidt, M. (2015). Solving the box-pushing problem using a spherical robot. pages 7–11.

Werger, B. and Mataric, M. J. (2000). Broadcast of local eligibility for multi-target observation. pages 347–356.

Willmann, J., Augugliaro, F., Cadalbert, T., D'Andrea, R., Gramazio, F., and Kohler, M. (2012). Aerial robotic construction towards a new field of architectural research. *International journal of architectural computing*, 10(3):439–459.

Wolkowski, R. and Lowery, B. (2008). Soil compaction: Causes, concerns, and cures. *University of Wisconsin*, pages 1–7.

Wooldridge, M. and Jennings, N. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.

Wu, M. H., Ogawa, S., and Konno, A. (2016a). Symmetry position/force hybrid control for cooperative object transportation using multiple humanoid robots. *Advanced Robotics*, 30(2):131–149.

Wu, Z., Hu, G., Feng, L., Wu, J., and Liu, S. (2016b). Collision avoidance for mobile robots based on artificial potential field and obstacle envelope modelling. *Assembly Automation*, 36(3).

Wurm, K. M., Stachniss, C., and Burgard, W. (2008). Coordinated multi-robot exploration using a segmentation of the environment. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1160–1165.

Yamada, S. and Saito, J. (1999). Adaptive action selection without explicit communication for multi-robot box-pushing. 3:1444–1449.

Yan, Z., Jouandeau, N., and Cherif, A. A. (2010). Sampling-based multi-robot exploration. pages 1–6.

Yan, Z., Jouandeau, N., and Cherif, A. A. (2012). Multi-robot heuristic goods transportation. pages 409–414.

Yan, Z., Jouandeau, N., and Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10:1–18.

Yang, J. and Dong, M. (2012). Design of a time-varying continuous control to solve the potential field local minima problem. *Applied Mechanics and Materials*, 130-134:2465–2469.

Zedadra, O., Seridi, H., Jouandeau, N., and Fortino, G. (2016). A cooperative switching algorithm for multi-agent foraging. *Engineering Applications of Artificial Intelligence*, 50:302–319.

Zhang and F. (2013). Detecting crop rows for automated rice transplanters based on radon transform. *Sensor Letters*, 11(6-7):1100–1105.

Zhang, C. and Noguchi, N. (2016). Cooperation of two robot tractors to improve work efficiency. *Adv Robot Autom*, 5(146):2.

Zhang, N., Wang, M., and Wang, N. (2002). Precision agriculturea worldwide overview. *Computers and Electronics in Agriculture*, 36(2-3):113–132.

Zheng, X., Jain, S., Koenig, S., and Kempe, D. (2005). Multi-robot forest coverage. pages 3852–3857.

Zheng, Z. and Tan, Y. (2015). Mobile target tracking of swarm robotics in unknown obstructive environment. *Handbook of Research on Design, Control, and Modeling of Swarm Robotics*, 26:397.

Zlot, R., Stentz, A., Dias, M. B., and Thayer, S. (2002). Market-driven multi-robot exploration.

Appendix A

Spraying Optimum Team Size

Referring to the total required time for spraying:

Given a field and the task of spraying, what is the most optimum team size with which spraying can be carried out as fast as possible?

$$t = \max_{1 \leq i \leq n} (t_{s_i} + t_{ta_i} + t_{st_i}) \tag{A.1}$$

Where:

t_{ta_i} is the task allocation time for robot i.

$t_{ta_i} = \frac{d_{\alpha l_1}}{v} + (2i-1)\tau + \frac{(\lambda+\epsilon)(i-1)}{v} + \sum_{j=2}^{i} \frac{d_{l_{j-1}l_j}}{v}$

t_{st_i} is the amount of time required for a robot to start its task.

$t_{st_i} = t_{ta_n} - t_{ta_i} + \tau_{init}(n-i)$

t_{s_i} is the time that takes for robot i to spray the allocated region.

$t_{s_i} = \frac{1}{v}(k_i l_p + (k_i - 1)d_f)$

In here, k_i is the number of tracks in a region, d_f is the distance between two consecutive regions, l_p is the length of a furrow, and v is the velocity of the robot.

Re-writing A.12:

$t = \max_{1 \leq i \leq n} (t_{s_i} + t_{ta_i} + t_{ta_n} - t_{ta_i} + \tau_{init}(n-i))$

$$t = \max_{1 \le i \le n} (t_{s_i} + t_{ta_n} + \tau_{init}(n-i))$$

$$t = \max_{1 \le i \le n} (\tfrac{1}{v}(k_i l_p + (k_i - 1)d_f) + \tfrac{d_{\alpha l_1}}{v} + (2n-1)\tau + \tfrac{(\lambda+\epsilon)(i-1)}{v} + \sum_{j=2}^{i} \tfrac{d_{l_{j-1}l_j}}{v} + \tau_{init}(n-i))$$

The total task allocation time for all robots is:

$$t_{ta_{tot}} = \frac{d_{\alpha l_1}}{v} + (2n-1)\tau + \frac{(\lambda+\epsilon)(n-1)}{v} + \sum_{i=2}^{n} \frac{d_{l_{j-1}l_j}}{v} \tag{A.2}$$

In here, $\sum_{i=2}^{n} \frac{d_{l_{j-1}l_j}}{v}$ refers to the width of the field W. Hence, A.2 can be re-written as:

$$t_{ta_{tot}} = \frac{d_{\alpha l_1}}{v} + (2n-1)\tau + \frac{(\lambda+\epsilon)(n-1)}{v} + \frac{W}{v} \tag{A.3}$$

Also, $W = K.d_f$, therefore:

$$t_{ta_{tot}} = \frac{d_{\alpha l_1}}{v} + (2n-1)\tau + \frac{(\lambda+\epsilon)(n-1)}{v} + \frac{K.d_f}{v} \tag{A.4}$$

Re-arranging A.4:

$$t_{ta_{tot}} = \frac{d_{\alpha l_1}}{v} + \frac{K.d_f}{v} - (\frac{(\lambda+\epsilon)}{v} + \tau) + (2\tau + \frac{(\lambda+\epsilon)}{v})n \tag{A.5}$$

We denote the constant $\frac{d_{\alpha l_1}}{v} + \frac{K.d_f}{v} - (\frac{(\lambda+\epsilon)}{v} + \tau)$ as C, and the coefficient by n as A, then Equation A.5 will be represented as:

$$t_{ta_{tot}} = C + An \tag{A.6}$$

Equation A.6 is a linear equation, and it is clear that with the number of robots the total allocation time with be increasing.

The total initiation time (until the moment the last robot start spraying) is $t_{ti_{tot}} =$

$\tau_{init}(n-1)$ or:

$$t_{ti_{tot}} = -\tau_{init} + \tau_{init} n \tag{A.7}$$

Equation A.7 also shows a simple linear relationship between the number of robots and total time for task initiation.

If we consider now the total spraying time, we can notice that part of this time comprises the total allocation time + time needed for spraying by the last robot. This time (spraying time for the last robot) is:

$$t_{s_n} = \frac{1}{v}(k_n l_p + (k_n - 1)d_f) \tag{A.8}$$

Bearing in mind that $k_n = \lceil \frac{K}{n} \rceil$, we can rewrite A.8 as follows:

$$t_{s_n} = \frac{1}{v}(\lceil \frac{K}{n} \rceil l_p + (\lceil \frac{K}{n} \rceil - 1)d_f) \tag{A.9}$$

Or

$$t_{s_n} = \frac{1}{v}(\lceil \frac{K}{n} \rceil (l_p + d_f)) - \frac{d_f}{v} \tag{A.10}$$

Note. We use $\lceil \frac{K}{n} \rceil$ (which is the smallest integer greater or equal to $\frac{K}{n}$), as this is the number of tracks assigned to the last robot, according to our construction.

The relation A.10 between time and the number of the robots is hyperbolic, and time decreases with the number of robots.

The total spraying time is, therefore,

$$t_{ts_{tot}} = t_{ti_{tot}} + t_{s_n} \tag{A.11}$$

And the total time for robots from the task allocation to completing spraying is:

$$t_{tot} = t_{ta_{tot}} + t_{ti_{tot}} + t_{s_n} \tag{A.12}$$

This equation consists of two increasing linear functions (which can be presented as one linear increasing function) and one decreasing hyperbolic function. Therefore the resulting function will have a global minimum. In order to determine this minimum, without the loss of generality, instead of integer n, we use real number x, $x > 0$. Then we rewrite A.10 as $t_{s_n} = \frac{1}{v}(\lceil \frac{K}{n} \rceil (l_p + d_f)) - \frac{d_f}{v}$ and re-arrange as:

$$t_{s_n} = \frac{K}{n}(l_p + d_f)\frac{1}{x} - \frac{d_f}{v} \tag{A.13}$$

Denoting $\frac{K}{v}(l_p + d_f)$ as B, and $\frac{d_f}{v}$ as D, we get

$t_{s_n} = B\frac{1}{x} - D$

Then, Equation A.12 takes the form of:

$$t_{tot} = C + Ax + \tau_{init}x - \tau_{init} + B\frac{1}{x} - D \tag{A.14}$$

Or

$$t_{tot} = (A + \tau_{init})x + B\frac{1}{x} + C - D - \tau_{init} \tag{A.15}$$

Differentiating A.15 with respect to x, and we get

$$t'_{tot} = (A + \tau_{init}) - B\frac{1}{x^2} \tag{A.16}$$

To find the extrema, we equate the right-hand side to zero, we get

$(A + \tau_{init}) - B\frac{1}{x^2} = 0$

As we interested only in positive x, we get the following value for x at the minimum:

$$x = \sqrt{\frac{B}{A + \tau_{init}}} \tag{A.17}$$

Substituting corresponding expressions for B and A into A.17:

$$x = \sqrt{\frac{\frac{K}{v}(l_p + d_f)}{2\tau + \frac{\lambda+\epsilon}{v} + \tau_{init}}} \tag{A.18}$$

and re-arranging it we have:

$$x = \sqrt{\frac{K(l_p + d_f)}{(2\tau + \tau_{init})v + \lambda + \epsilon}} \tag{A.19}$$

Appendix B

C++ Code for Artificial Potential Field Using ROS

```
#include <ros/ros.h>
#include <sensor_msgs/LaserScan.h>
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Twist.h>
#include <tf/tf.h>
#include <geometry_msgs/Quaternion.h>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <string>
#include <sstream>

using namespace std;

#define PI 3.14159265359
#define Rad2Deg 57.2957795
#define Deg2Rad 0.0174532925
#define PITCH 0.3522504892367
#define DistanceTh 0.2
#define DistanceCrt 1.5
#define DistanceCrtTwo 0.5
#define MaxVel 0.5

void stop(void);
struct point
{
 float x;
 float y;
 float theta;
};
struct force
{
 float mag;
 float theta;
 float x;
 float y;
};
point target, current;

bool targetIsReached = false;

ros::Publisher vel_pub;
geometry_msgs::Twist vel;
int robot_id = 0;
class APFClass
{
```

```
47  public:
48   APFClass();
49  private:
50   ros::NodeHandle nh;
51
52   ros::Subscriber laser_sub;
53   ros::Subscriber pose_sub;
54
55   void laserCB(const sensor_msgs::LaserScan::ConstPtr &scan);
56   void poseCB(const nav_msgs::Odometry::ConstPtr &pose);
57  };
58
59  APFClass::APFClass()
60  {
61   ros::NodeHandle n("~");
62   n.getParam("robot_id",robot_id);
63   ostringstream ss;
64   ss << "robot_";
65   ss << robot_id;
66   string laser_pipename = ss.str() + "/base_scan";
67   string pose_pipename = ss.str() + "/base_pose_ground_truth";
68   string velcmd_pipename = ss.str() + "/cmd_vel";
69
70   laser_sub = nh.subscribe<sensor_msgs::LaserScan>
71         (laser_pipename,10,&APFClass::laserCB,this);
72   pose_sub = nh.subscribe<nav_msgs::Odometry>
73         (pose_pipename,10,&APFClass::poseCB,this);
74   vel_pub = nh.advertise<geometry_msgs::Twist>
75        (velcmd_pipename,10);
76   target.x = 0.0;
77   target.y = 0.0;
78  }
79
80  void APFClass::laserCB
81        (const sensor_msgs::LaserScan::ConstPtr &scan)
82  {
83   float beams[512];
84   point dir, obj_new[512], obj[512];
85   force attraction, repulsion[512], sum;
86   int RotCoef = 1;
87   sum.x = 0.0;
88   sum.y = 0.0;
89   int beamCounter = 1;
90   dir.x = target.x - current.x;
91   dir.y = target.y - current.y;
92   dir.theta = atan2(dir.y,dir.x)*Rad2Deg;
93   float distanceToTarget = sqrt(dir.x*dir.x + dir.y*dir.y);
94   float R[2][2] = {{cos((90-current.theta)*Deg2Rad),
95             -1*sin((90-current.theta)*Deg2Rad)},
96            {sin((90-current.theta)*Deg2Rad),
97             cos((90-current.theta)*Deg2Rad)}};
98   if (distanceToTarget >= DistanceTh)
99   {
100   for (int i=0; i<=511; i++)
101   {
102    if (scan->ranges[i] < 4.5 )
103    {
104     beams[i] = scan->ranges[i];
105
106     obj[i].x = beams[i]*cos((i*PITCH)*Deg2Rad);
```

```
107       obj[i].y = beams[i]*sin((i*PITCH)*Deg2Rad);
108       beamCounter++;
109      }
110      else if (scan->ranges[i] == 4.5)
111      {
112       beams[i] = 0.0;
113       obj[i].x = 0.0;
114       obj[i].y = 0.0;
115      }
116     }
117
118     //form the orientation matrix
119     for (int i=0; i<= 511; i++)
120     {
121      if (beams[i] > 0.0 && beams[i] < 4.5)
122      {
123
124       repulsion[i].x = obj[i].x;
125       repulsion[i].y = obj[i].y;
126
127       //remember: repulsion force is 180 degrees mirrored of the
128       //vector describing the object and the robot
129       if (beams[i] <= DistanceCrt && beams[i] > DistanceCrtTwo)
130       {
131        repulsion[i].x = 20*obj[i].x;
132        repulsion[i].y = 20*obj[i].y;
133        RotCoef = 10;
134       }
135       if (beams[i] <= DistanceCrtTwo)
136       {
137        repulsion[i].x = 100*obj[i].x;
138        repulsion[i].y = 100*obj[i].y;
139        RotCoef = 100;
140       }
141       if (beams[i] > DistanceCrt)
142       {
143        repulsion[i].x = obj[i].x;
144        repulsion[i].y = obj[i].y;
145        RotCoef = 1;
146       }
147       repulsion[i].mag = 1/sqrt(repulsion[i].x*repulsion[i].x
148                 +repulsion[i].y*repulsion[i].y);
149       repulsion[i].theta = atan2(repulsion[i].y,repulsion[i].x);
150      }
151      else if (beams[i] == 0.0)
152      {
153       repulsion[i].x = 0.0;
154       repulsion[i].y = 0.0;
155       repulsion[i].mag = 0.0;
156       repulsion[i].theta = 0.0;
157      }
158
159      sum.x = sum.x + repulsion[i].x;
160      sum.y = sum.y + repulsion[i].y;
161     }
162
163     float rx = sum.x*cos(180*Deg2Rad) - sum.y*sin(180*Deg2Rad);
164     float ry = sum.x*sin(180*Deg2Rad) + sum.y*cos(180*Deg2Rad);
165     sum.x = rx*R[0][0] - ry*R[0][1];
166     sum.y = -rx*R[1][0] + ry*R[1][1];
```

```
167     sum.mag = sqrt(sum.x*sum.x+sum.y*sum.y);
168     sum.theta = Rad2Deg*atan2(sum.y,sum.x);
169
170     //applying robot's orientation
171
172
173     //attraction force
174     attraction.mag = sqrt(dir.x*dir.x+dir.y*dir.y);
175     attraction.theta = Rad2Deg*atan2(dir.y,dir.x);
176     attraction.x = dir.x;
177     attraction.y = dir.y;
178
179     //sum force
180     int Krep = 7;
181     int Katt = 20;
182     sum.x = sum.x + Katt*attraction.x;
183     sum.y = sum.y + Katt*attraction.y;
184     sum.theta = Rad2Deg*atan2(sum.y,sum.x);
185     sum.mag = sqrt(sum.x*sum.x+sum.y*sum.y);
186
187     /*Here is where you assign speed to the motors*/
188     if (abs(sum.theta-current.theta) < 180)
189     {
190       vel.angular.z = RotCoef*0.02*(sum.theta-current.theta);
191     }
192     else if ((sum.theta-current.theta > 180)
193         || (sum.theta-current.theta < -180))
194     {
195       vel.angular.z = RotCoef*-0.02*(sum.theta-current.theta);
196     }
197     vel.linear.x = sum.mag;
198     //limiting the linear speed to 0.75 m/s
199     if (vel.linear.x >= MaxVel)
200     {
201       vel.linear.x = MaxVel;
202     }
203     if (vel.linear.x <= -1*MaxVel)
204     {
205       vel.linear.x = -1*MaxVel;
206     }
207     vel_pub.publish(vel);
208
209   }
210   else if (distanceToTarget < DistanceTh)
211   {
212     stop();
213   }
214 }
215 void APFClass::poseCB
216     (const nav_msgs::Odometry::ConstPtr &pose)
217 {
218   current.x = pose->pose.pose.position.x;
219   current.y = pose->pose.pose.position.y;
220   current.theta = Rad2Deg *
221     tf::getYaw(pose->pose.pose.orientation);
222 }
223 void stop(void)
224 {
225   vel.linear.x = 0.0;
226   vel.linear.y = 0.0;
```

```
227   vel.linear.z = 0.0;
228   vel.angular.x = 0.0;
229   vel.angular.y = 0.0;
230   vel.angular.z = 0.0;
231   vel_pub.publish(vel);
232  }
233  int main(int argc, char** argv)
234  {
235   ros::init(argc,argv,"mrs_apf");
236   APFClass apfc;
237   ros::spin();
238   return 0;
239  }
```

Appendix C

C++ Code for Cluster Class

```
#include<ros/ros.h>
#include<cluster_extractor/cluster.h>
#include<sensor_msgs/PointCloud.h>
#include<utility>
#include<math.h>
#include<vector>

using namespace std;


 Cluster::Cluster()
 {

 }
 Cluster::~Cluster()
 {

 }
 void Cluster::addMember(geometry_msgs::Point32 p)
 {
  members.points.push_back(p);
  sum_x += p.x;
  sum_y += p.y;
  sum_x2 += p.x * p.x;
  sum_y2 += p.y * p.y;
  sum_xy += p.x * p.y;
 }
 void Cluster::getMembers(sensor_msgs::PointCloud &p)
 {
  p = members;
 }
 size_t Cluster::size()
 {
  return members.points.size();
 }
 pair<double, double> Cluster::getTrend()
 {
  x_bar = sum_x / members.points.size();
  y_bar = sum_y / members.points.size();

  slope = (sum_xy - y_bar * sum_x - x_bar * sum_y +
      members.points.size() * x_bar * y_bar)
      / (sum_x2 - 2 * x_bar * sum_x + members.points.size()
      * x_bar * x_bar);
  y_intercept = y_bar - slope * x_bar;

```

```
47    angle_with_x_axis = (atan(slope) * 180.0 / M_PI) + 90.0;
48    return std::make_pair(slope, y_intercept);
49  }
50  void Cluster::clear()
51  {
52    members.points.clear();
53    slope = 0.0;
54    y_intercept = 0.0;
55    x_bar = 0.0;
56    y_bar = 0.0;
57    sum_x = 0.0;
58    sum_y = 0.0;
59    sum_x2 = 0.0;
60    sum_xy = 0.0;
61    leftBound = 0;
62    rightBound = 0;
63    distanceToLeftCluster = 0.0;
64    distanceToRightCluster = 0.0;
65  }
66  double Cluster::getSlope()
67  {
68    return slope;
69  }
70  double Cluster::getYIntercept()
71  {
72    return y_intercept;
73  }
74  double Cluster::getAngleWithXAxis()
75  {
76    return angle_with_x_axis;
77  }
```

Appendix D

C++ Code for Cluster Finding

```
1  #include<ros/ros.h>
2  #include<sensor_msgs/LaserScan.h>
3  #include<sensor_msgs/PointCloud.h>
4  #include<geometry_msgs/Point32.h>
5  #include<cluster_extractor/LaserClusters.h>
6
7  #include<math.h>
8  #include<utility>
9
10 #include<cluster_extractor/cluster.h>
11 #include<cluster_extractor/local.h>
12 #include<cluster_extractor/laser_analysis.h>
13 using namespace std;
14
15
16  void laser_analysis::rectify(const sensor_msgs::LaserScan
17  &src, sensor_msgs::LaserScan &des)
18  {
19   des = src;
20   for(int i = 0; i < des.ranges.size(); i++)
21   {
22    if(des.ranges[i] < des.range_min)
23    {
24     des.ranges[i] = des.range_min;
25    }
26    else if(des.ranges[i] > des.range_max)
27    {
28     des.ranges[i] = des.range_max;
29    }
30   }
31  }
32
33  int laser_analysis::getIndex(double angular_res, double beam_angle)
34  {
35   return ((beam_angle  +  45.0) / angular_res);
36  }
37  double laser_analysis::getAngle(double angular_res, int beam_index)
38  {
39   return ((beam_index * angular_res )- 45.0);
40  }
41  pair<int, int> laser_analysis::getBoundaries
42  (const sensor_msgs::LaserScan &src, double frontFieldOfViewAngle)
43  {
44   double min_angle = (src.angle_min * 180.0 / M_PI);
45   double max_angle = (src.angle_max * 180.0 / M_PI);
46   double angle_increment = src.angle_increment * 180.0 / M_PI;
```

```
47    double correctAng = (max_angle - min_angle) - 180.0;
48
49    return make_pair(getIndex(angle_increment,
50    (correctAng + (frontFieldOfViewAngle / 2 ))),
51    getIndex(angle_increment,
52    (correctAng - (frontFieldOfViewAngle / 2))));
53   }
54   void laser_analysis::convertLaserScanToPointCloud
55   (const sensor_msgs::LaserScan &l, sensor_msgs::PointCloud &pc)
56   {
57    pc.header = l.header;
58
59    int range_size = (int)l.ranges.size();
60    double angle_increment = l.angle_increment;
61    double angle_min = l.angle_min;
62    double max_range =l.range_max;
63    double min_range = l.range_min;
64
65    for (int i = 0; i < l.ranges.size(); i++)
66    {
67     if(l.ranges[i] >= min_range && l.ranges[i] <= max_range)
68     {
69      geometry_msgs::Point32 p;
70      p.x = l.ranges[i] * cos((i * angle_increment) + angle_min);
71      p.y = l.ranges[i] * sin((i * angle_increment) + angle_min);
72      p.z = 0.0;
73
74      pc.points.push_back(p);
75     }
76    }
77   }
78   void laser_analysis::convertLaserScanToPointCloud
79   (const sensor_msgs::LaserScan &l, sensor_msgs::PointCloud &pc,
80   bool rangeFilterIsOn, double maxRange, double front_field_of_view)
81   {
82    pc.header = l.header;
83
84    pair<int, int> boundaries = getBoundaries(l, front_field_of_view);
85
86    int range_size = (int)l.ranges.size();
87    double angle_increment = l.angle_increment;
88    double angle_min = l.angle_min;
89
90    double max_range = 0.0;
91    double min_range = l.range_min;
92    if(rangeFilterIsOn)
93    {
94     max_range = maxRange;
95    }
96    else
97    {
98     max_range =  l.range_max;
99    }
100
101   for (int i = boundaries.second; i <= boundaries.first; i++)
102   {
103    if(l.ranges[i] >= min_range && l.ranges[i] <= max_range)
104    {
105     geometry_msgs::Point32 p;
106     p.x = l.ranges[i] * cos((i * angle_increment) + angle_min);
```

```
107       p.y = l.ranges[i] * sin((i * angle_increment) + angle_min);
108       p.z = 0.0;
109
110       pc.points.push_back(p);
111      }
112     }
113    }
114
115    void laser_analysis::getClusters
116    (const sensor_msgs::PointCloud &src, vector<Cluster> &res,
117    double max_cluster_distance, int min_cluster_size)
118    {
119     Cluster l_cluster;
120     l_cluster.distanceToRightCluster = 0.0;
121     for(int i = 0; i < src.points.size() - 1; i++)
122     {
123      geometry_msgs::Point32 p = src.points[i];
124      geometry_msgs::Point32 pn = src.points[i+1];
125      double dx = p.x - pn.x;
126      double dy = p.y - pn.y;
127      double distanceToNeighbour = sqrt(dx*dx + dy*dy);
128      if(distanceToNeighbour < max_cluster_distance)
129      {
130       //This is the right index
131       if(l_cluster.size() == 0)
132       {
133        l_cluster.rightBound = i;
134        l_cluster.addMember(p);
135       }
136       if(i == src.points.size() - 2 )
137       {
138        l_cluster.leftBound = i;
139        l_cluster.distanceToLeftCluster = 0.0;
140        l_cluster.getTrend();
141        l_cluster.addMember(p);
142        if(l_cluster.size() >= min_cluster_size)
143        {
144         res.push_back(l_cluster);
145        }
146
147       }
148       else
149       {
150        l_cluster.addMember(p);
151       }
152      }
153      else
154      {
155       l_cluster.leftBound = i;
156       l_cluster.addMember(p);
157       l_cluster.distanceToLeftCluster = distanceToNeighbour;
158       l_cluster.getTrend();
159
160       if(l_cluster.size() >= min_cluster_size)
161       {
162        res.push_back(l_cluster);
163       }
164       //reseting the local cluster
165       l_cluster.clear();
166       l_cluster.rightBound = i+1;
```

```
167       l_cluster.distanceToRightCluster = distanceToNeighbour;
168      }
169     }
170    }
171    void laser_analysis::clustersSizeFilter
172    (vector<Cluster> &src, vector<Cluster> &des, int size_th)
173    {
174     for(int i = 0; i < src.size(); i++)
175     {
176      if((int)src[i].size() >= size_th)
177      {
178       des.push_back(src[i]);
179      }
180     }
181    }
182    void laser_analysis::getLargestCluster
183    (vector<Cluster> &src, Cluster &des)
184    {
185     int size = 0;
186     int index = 0;
187     for(int i = 0; i < src.size(); i++)
188     {
189      if((int)src[i].size() > size)
190      {
191       index = i;
192      }
193     }
194     des = src[index];
195    }
196    void laser_analysis::publishClusters
197    (vector<Cluster> &src, ros::Publisher &pub,
198    const string &frame_id, int seq,
199    double max_cluster_distance, int min_cluster_size)
200    {
201     cluster_extractor::LaserClusters lcs;
202     lcs.header.frame_id =frame_id;
203     lcs.header.seq = seq;
204     lcs.header.stamp = ros::Time::now();
205     lcs.max_cluster_distance = max_cluster_distance;
206     lcs.min_cluster_size = min_cluster_size;
207
208     for(int i = 0; i < src.size(); i++)
209     {
210      cluster_extractor::LaserCluster lc;
211      sensor_msgs::PointCloud p;
212      src[i].getMembers(p);
213      lc.points = p.points;
214      lc.rightBound = src[i].rightBound;
215      lc.leftBound = src[i].leftBound;
216      lc.distanceToLeftCluster = src[i].distanceToLeftCluster;
217      lc.distanceToRightCluster = src[i].distanceToRightCluster;
218      lc.slope = src[i].getSlope();
219      lc.y_intercept = src[i].getYIntercept();
220      lc.clusterIndex = i;
221      lc.angle_with_x_axis = src[i].getAngleWithXAxis();
222
223      lcs.clusters.push_back(lc);
224     }
225     pub.publish(lcs);
226    }
```

```
227  void laser_analysis::searchForLocals(const sensor_msgs::PointCloud &src,
228  vector<Local> &des, int n_search, bool searchForMinima)
229  {
230   for(int cnt = n_search + 1; cnt < src.points.size() - n_search; cnt++)
231   {
232     vector<geometry_msgs::Point32> temp;
233     bool isItLocal = true;
234     geometry_msgs::Point32 p = src.points[cnt];
235     double mag = sqrt((p.x)*(p.x) + (p.y)*(p.y));
236
237     int lowBound = cnt - (n_search / 2);
238     int highBound = cnt + (n_search / 2);
239
240     for(int i = lowBound; i <= highBound; i++)
241     {
242      if(i == cnt) continue;
243
244      geometry_msgs::Point32 p_i = src.points[i];
245      temp.push_back(p_i);
246      double nmag = sqrt((p_i.x)*(p_i.x) + (p_i.y)*(p_i.y));
247      if(searchForMinima && mag > nmag)
248      {
249       isItLocal = false;
250       break;
251      }
252      if(!searchForMinima && mag < nmag)
253      {
254       isItLocal = false;
255       break;
256      }
257     }
258     if(isItLocal )
259     {
260      Local lm;
261      lm.setLocal(temp, searchForMinima);
262      des.push_back(lm);
263     }
264   }
265  }
266  void laser_analysis::searchForLocals(Cluster &src,
267  vector<Local> &des, int n_search, bool searchForMinima)
268  {
269   sensor_msgs::PointCloud members;
270   src.getMembers(members);
271   for(int cnt = n_search + 1; cnt < (int) src.size() - n_search; cnt++)
272   {
273    vector<geometry_msgs::Point32> temp;
274    bool isItLocal = true;
275    geometry_msgs::Point32 p = members.points[cnt];
276    double mag = sqrt((p.x)*(p.x) + (p.y)*(p.y));
277
278
279    int lowBound = cnt - (n_search / 2);
280    int highBound = cnt + (n_search / 2);
281    for(int i = lowBound; i <= highBound; i++)
282    {
283     if(i == cnt) continue;
284     geometry_msgs::Point32 p_i = members.points[i];
285     temp.push_back(p_i);
286     double nmag = sqrt((p_i.x)*(p_i.x) + (p_i.y)*(p_i.y));
```

```
287       if(searchForMinima && mag > nmag)
288       {
289        isItLocal = false;
290        break;
291       }
292       if(!searchForMinima && mag < nmag)
293       {
294        isItLocal = false;
295        break;
296       }
297      }
298      if(isItLocal )
299      {
300       Local lm;
301       lm.setLocal(temp, searchForMinima);
302       des.push_back(lm);
303      }
304     }
305    }
306    void laser_analysis::AverageMagnitudeFilter
307    (vector<Local> &src, vector<Local> &res, double avg_coef_limit)
308    {
309     double avg = 0.0;
310     for(Local local : src)
311     {
312      avg += local.getMagnitude();
313     }
314     avg = avg / src.size();
315
316     for(int i = 0; i < src.size(); i++)
317     {
318      if(src[i].getMagnitude() >= avg *avg_coef_limit)
319      {
320       res.push_back(src[i]);
321      }
322     }
323    }
324    void laser_analysis::neighbourDistanceFilter
325    (vector<Local> &src, vector<Local> &res,
326    double ThresholdDistance, double Error_Th)
327    {
328     double sm_th = 0.1;
329     for (int i = 0; i < src.size(); i++)
330     {
331      int inRangeNeighbours = 0;
332      for(int j = 0; j < src.size(); j++)
333      {
334       if(i == j) continue;
335
336       double n_dist = abs(src[i].centre.y - src[j].centre.y);
337       //double n_dist = abs(p.points[src[i].pc_index].y - p.points[src[j].pc_index].y);
338       if((n_dist <= ThresholdDistance + Error_Th
339        &&  n_dist >= ThresholdDistance - Error_Th) ||
340         (n_dist <= 2*(ThresholdDistance + Error_Th)
341         &&  n_dist >= 2*(ThresholdDistance - Error_Th)))
342       {
343        inRangeNeighbours++;
344       }
345      }
346      if(inRangeNeighbours >= 1)
```

```
347     {
348       src[i].inRangeNeighbours = inRangeNeighbours;
349       res.push_back(src[i]);
350     }
351    }
352   }
353  //}
```

Appendix E

C++ Code for Chilitag Fiducial Finding

```
1  #include<iostream>
2  #include<string>
3  #include<stdio.h> //for popen();
4  #include<sstream>
5
6  //ROS includes
7  #include<sensor_msgs/Image.h>
8  #include<sensor_msgs/image_encodings.h>
9  #include<image_transport/image_transport.h>
10 #include<cv_bridge/cv_bridge.h>
11 #include<ros/ros.h>
12 #include <brain_box_msgs/TargetPoseArray.h>
13 #include<brain_box_msgs/TargetPose.h> //for publishing fiducial locations
14 #include<brain_box_msgs/Point2.h>
15
16 //opencv includes
17 #include<opencv2/opencv.hpp>
18 #include<opencv2/highgui.hpp>
19 #include<opencv2/imgproc.hpp>
20 #include<opencv2/imgcodecs.hpp>
21
22 //third party library includes
23 #include<chilitags.hpp>
24
25 using namespace std;
26 using namespace cv;
27
28 struct gui_component
29 {
30  int max;
31  int handler;
32  float global;
33 };
34
35 //MODE BOOL
36 bool readCamera = false;
37
38 bool running = true;
39
40 //thread handler
41 pthread_t frame_collecting_thread;
42
43 // First, we set up some constants related to the information overlaid
44 // on the captured image
45 const static cv::Scalar COLOR(255, 0, 255);
46 // OpenCv can draw with sub-pixel precision with fixed point coordinates
```

```
47  static const int SHIFT = 16;
48  static const float PRECISION = 1<<SHIFT;
49  int native_image_width_;
50  int native_image_height_;
51  int device = -1;
52
53  gui_component brightness, contrast, sharpen, gray, frame_control;
54
55  ros::Publisher pos_pub;
56
57  //this function gets the cv source frame and
58  returns available chilitags on the screen
59  inline void chilitags_detect(cv::Mat &src,
60  chilitags::TagCornerMap &tags)
61  {
62   // The tag detection happens in the Chilitags class.
63   chilitags::Chilitags chilitags;
64   /* The detection is not perfect,
65     so if a tag is not detected during one frame,
66      the tag will shortly disappears, which results in flickering.
67      To address this, Chilitags "cheats" by keeping tags for n frames
68      at the same position. When tags disappear for more than 5 frames,
69     Chilitags actually removes it.
70     Here, we cancel this to show the raw detection results.
71   */
72      chilitags.setFilter(0, 0.0f);
73      // Detect tags on the current image;
74      // The resulting map associates tag ids (between 0 and 1023)
75      // to four 2D points corresponding to the corners positions
76      // in the picture.
77      tags = chilitags.find(src);
78  }
79
80  /*
81   a function to change brightness and contrast of the image
82   new_image(i,j) = brightness_coef * old_image(i,j) + contrast_coef
83  */
84
85
86  void ZEDimgCB(const sensor_msgs::ImageConstPtr &img)
87  {
88   brain_box_msgs::TargetPoseArray bbPoses_;
89   cv_bridge::CvImagePtr cv_ptr;
90   try
91   {
92    cv_ptr = cv_bridge::toCvCopy(img, sensor_msgs::image_encodings::BGR8);
93    cv::Mat output_image = cv_ptr->image;
94    chilitags::TagCornerMap tags;
95    //actual detection
96    chilitags_detect(output_image, tags);
97    for (const std::pair<int, chilitags::Quad> & tag : tags)
98    {
99     brain_box_msgs::TargetPose tagPose;
100    tagPose.target_id = tag.first;
101       int id = tag.first;
102       // We wrap the corner matrix into a datastructure that allows an
103       // easy access to the coordinates
104       const cv::Mat_<cv::Point2f> corners(tag.second);
105       // We start by drawing the borders of the tag
106       for (size_t i = 0; i < 4; ++i)
```

```
107     {
108      brain_box_msgs::Point2 bbPoint;
109      bbPoint.x = corners(i).x;
110      bbPoint.y = corners(i).y;
111      tagPose.outline.push_back(bbPoint);
112            cv::line(
113                output_image,
114                PRECISION*corners(i),
115                PRECISION*corners((i+1)%4),
116      #ifdef OPENCV3
117                 COLOR, 1, cv::LINE_AA, SHIFT);
118      #else
119                 COLOR, 1, CV_AA, SHIFT);
120      #endif
121        }
122        cv::Point2f center = 0.5f*(corners(0) + corners(2));
123        cv::putText(output_image, cv::format("%d", id),
124      center, cv::FONT_HERSHEY_SIMPLEX, 1.0f, COLOR);
125     //ROS_INFO("Detected ID: %d",id);
126     bbPoses_.poses.push_back(tagPose);
127    }
128    pos_pub.publish(bbPoses_);
129    imshow("chilitags",output_image);
130
131   }
132   catch (cv_bridge::Exception& e)
133   {
134    ROS_ERROR("cv_bridge exception: %s", e.what());
135    return;
136   }
137   if(waitKey(1) >= 0) ; //Do not act when a key is pressed as SIGINT is being caught.
138 }
139
140 int main(int argc, char** argv)
141 {
142  ros::init(argc,argv,"chilitags_zed_subscribe");
143
144  ros::NodeHandle nh_;
145  pos_pub = nh_.advertise<brain_box_msgs::TargetPoseArray>
146   ("/chilitags/position",100);
147  image_transport::ImageTransport it_(nh_);
148   image_transport::Subscriber img_sub =
149   it_.subscribe("/left/image_rect_color",10, &ZEDimgCB);
150
151  namedWindow("chilitags",WINDOW_NORMAL);//creating a window for display
152  ros::spin();
153  return 0;
154 }
```

Appendix F

C++ Code for Color-based Pattern Recognition

```
1  #include <ros/ros.h>
2  #include <image_transport/image_transport.h>
3  #include <cv_bridge/cv_bridge.h>
4  #include <sensor_msgs/image_encodings.h>
5  #include <opencv2/imgproc/imgproc.hpp>
6  #include <opencv2/highgui/highgui.hpp>
7  #include <string>
8  #include <tinyxml.h>
9  #include <math.h>
10 #include <iostream>
11 #include <vector>
12 #include <geometry_msgs/Pose2D.h>
13
14 #define PI 3.14159265
15 #define FltTh 1
16
17 using namespace std;
18 using namespace cv;
19
20 struct ColorParam
21 {
22  int lHue;
23  int hHue;
24  int lSat;
25  int hSat;
26  int lVal;
27  int hVal;
28  string fMode;
29  int erode;
30  int dilate;
31 };
32
33 struct Blob
34 {
35  int x;
36  int y;
37  int height;
38  int width;
39  double area;
40  float x_c;
41  float y_c;
42 };
43
44 /*struct Pose
45 {
46  float x;
```

```
 float y;
 float th;
};*/
geometry_msgs::Pose2D robot_pose;
//Pose robot_pose;
Blob sb,bb;
ColorParam cp;
int medianCount = 0;
double t_time_p;
int freq = 0;
double headingSum;
double c_x_sum, c_y_sum;
int loadColorParam(string COLORCODE);
double StableData(vector<double> raw);
vector<double> raw(FltTh);
class ImageConverter
{
    ros::NodeHandle nh_;
    image_transport::ImageTransport it_;
    image_transport::Subscriber image_sub_;
  ros::Publisher pos_publisher;
public:
    ImageConverter(): it_(nh_)
    {
        //Subscrive to input video feed and publish output video feed
        image_sub_ = it_.subscribe
    ("/usb_cam/image_raw", 1,&ImageConverter::imageCb, this);
    pos_publisher = nh_.advertise<geometry_msgs::Pose2D>("/RED/pose",5);
    }
    ~ImageConverter()
    {
    }
    void imageCb(const sensor_msgs::ImageConstPtr& msg)
    {
    ros::Time newTime = ros::Time::now();
    double t_time = newTime.sec + double(newTime.nsec)/1000000000;
    //ROS_INFO("Now = %f" ,t_time);
        cv_bridge::CvImagePtr cv_ptr;
        /*1. convert the ros_image to opencv_image*/
        try
        {
            cv_ptr = cv_bridge::toCvCopy
      (msg, sensor_msgs::image_encodings::BGR8);
        }
        catch (cv_bridge::Exception& e)
        {
            ROS_ERROR("cv_bridge exception: %s", e.what());
            return;
        }
    Mat hsv,img,thresh;
    img = cv_ptr->image;

    //Convert the captured frame from BGR to HSV
    cvtColor(img, hsv, COLOR_BGR2HSV);
    //Extract Threshold based on the appropriate color range
    inRange(hsv, Scalar(cp.lHue, cp.lSat, cp.lVal),
    Scalar(cp.hHue, cp.hSat, cp.hVal), thresh);

    //Apply the selected Morphological filter
    if (cp.fMode.compare("open") == 0)
```

```
107     {
108         if (cp.erode != 0)
109         {
110          erode(thresh, thresh,
111          getStructuringElement(MORPH_ELLIPSE,
112          Size(cp.erode, cp.erode)));
113         }
114         if (cp.dilate != 0)
115         {
116          dilate( thresh, thresh, getStructuringElement
117          (MORPH_ELLIPSE, Size(cp.dilate, cp.dilate)));
118         }
119     }
120     if (cp.fMode.compare("close") == 0)
121     {
122         if (cp.dilate != 0)
123         {
124          dilate( thresh, thresh, getStructuringElement
125          (MORPH_ELLIPSE, Size(cp.dilate, cp.dilate)));
126         }
127         if (cp.erode != 0)
128         {
129          erode(thresh, thresh, getStructuringElement
130          (MORPH_ELLIPSE, Size(cp.erode, cp.erode)));
131         }
132     }
133
134     //Apply Contour to the obtained threshold
135     Mat thresh_clone = thresh.clone();
136     std::vector<std::vector<cv::Point> > contours;
137     std::vector<cv::Vec4i> hierarchy;
138     cv::findContours( thresh_clone,
139                       contours,
140                       hierarchy,
141                       CV_RETR_TREE,
142                       CV_CHAIN_APPROX_SIMPLE,
143                       cv::Point(0, 0));
144     std::vector<std::vector<cv::Point> > contours_poly( contours.size() );
145     std::vector<cv::Rect> boundRect(contours.size());
146
147     //Filter 1: There must be only two blobs in the frame.
148     if (contours.size() == 2)
149     {
150      for( int i = 0; i < contours.size(); i++ )
151        {
152            cv::approxPolyDP( cv::Mat(contours[i]), contours_poly[i], 3, true);
153            boundRect[i] = cv::boundingRect(cv::Mat(contours_poly[i]));
154        }
155      //Sorting blobs: Compare areas - contour changes order of detection
156      if(boundRect[0].area() < boundRect[1].area())
157      {
158       bb.x = boundRect[1].x;
159       bb.y = boundRect[1].y;
160       bb.area = boundRect[1].area();
161       bb.height = boundRect[1].height;
162       bb.width = boundRect[1].width;
163
164       sb.x = boundRect[0].x;
165       sb.y = boundRect[0].y;
166       sb.area = boundRect[0].area();
```

```
167         sb.height = boundRect[0].height;
168         sb.width = boundRect[0].width;
169        }
170        else if(boundRect[0].area() > boundRect[1].area())
171        {
172         sb.x = boundRect[1].x;
173         sb.y = boundRect[1].y;
174         sb.area = boundRect[1].area();
175         sb.height = boundRect[1].height;
176         sb.width = boundRect[1].width;
177
178         bb.x = boundRect[0].x;
179         bb.y = boundRect[0].y;
180         bb.area = boundRect[0].area();
181         bb.height = boundRect[0].height;
182         bb.width = boundRect[0].width;
183        }
184
185        //Calculating the centre of the blobs
186        bb.x_c = floor(bb.x + bb.width/2);
187        bb.y_c = floor(bb.y + bb.height/2);
188
189        sb.x_c = floor(sb.x + sb.width/2);
190        sb.y_c = floor(sb.y + sb.height/2);
191
192        //Obtaining the center of the robot
193        Moments oMoments = moments(thresh);
194        robot_pose.x = floor(oMoments.m10/oMoments.m00);
195        robot_pose.y = floor(oMoments.m01/oMoments.m00);
196        robot_pose.theta = atan2((int)(sb.y-bb.y),(int)(sb.x-bb.x))*180/PI;
197
198        pos_publisher.publish(robot_pose);
199        freq++;
200       }
201
202       if (t_time - t_time_p >= 1.0)
203       {
204        ROS_INFO("Frequency = %d",freq);
205        t_time_p = t_time;
206        freq = 0;
207       }
208        imshow("Image",thresh);
209           if (waitKey(30) == 27)
210           {
211               cout << "esc key is pressed by user" << endl;
212           }
213       }
214  };
215
216  double StableData(vector<double> raw)
217  {
218   double res = 0.0;
219   int max = 0;
220   int temp[10];
221   //1. search
222   for (int i=0; i<raw.size(); i++)
223   {
224    temp[i] = 0;
225    for (int j=0; j<raw.size(); j++)
226    {
```

```
227     if (raw[i] == raw[j])
228     {
229      temp[i]++;
230     }
231    }
232   }
233   //2. find max
234   for (int i=0; i<raw.size(); i++)
235   {
236    if (temp[i] > max)
237    {
238     res = raw[i];
239     max = temp[i];
240    }
241   }
242   return res;
243  }
244
245  int loadColorParam(string COLORCODE)
246  {
247    TiXmlDocument doc
248   ("/home/robot/catkin_ws/src/localisation_system/src/image_param.xml");
249   if (!doc.LoadFile())
250   {
251    cerr << doc.ErrorDesc() << endl;
252    return -1;
253   }
254   TiXmlElement* color = doc.FirstChildElement();
255   if (color == NULL) return -1;
256   for(TiXmlElement* elem = color->FirstChildElement();
257    elem != NULL; elem = elem->NextSiblingElement())
258   {
259      string elemName = elem->Value();
260    const char* attr;
261    if (elemName == COLORCODE.c_str())
262    {
263     elem->Attribute("lHue",&cp.lHue);
264     elem->Attribute("hHue",&cp.hHue);
265     elem->Attribute("lSat",&cp.lSat);
266     elem->Attribute("hSat",&cp.hSat);
267     elem->Attribute("lVal",&cp.lVal);
268     elem->Attribute("hVal",&cp.hVal);
269     cp.fMode = elem->Attribute("mode");
270     elem->Attribute("erode",&cp.erode);
271     elem->Attribute("dilate",&cp.dilate);
272     return 1;
273    }
274   }
275  }
276
277  int main(int argc, char** argv)
278  {
279   int parsingCheck = loadColorParam("RED");
280   if (parsingCheck == 1)
281   {
282    ROS_INFO("HUE: low = %d, high = %d",cp.lHue,cp.hHue);
283    ROS_INFO("SAT: low = %d, high = %d",cp.lSat,cp.hSat);
284    ROS_INFO("VAL: low = %d, high = %d",cp.lVal,cp.hVal);
285    ROS_INFO("Filter: mode = %s, erode = %d, dilate = %d",
286    cp.fMode.c_str(),cp.erode,cp.dilate);
```

```
287   }
288   else
289   {
290    ROS_INFO("Problem␣parsing␣paramters.␣Check␣image_param.xml");
291    return EXIT_FAILURE;
292   }
293   namedWindow("Image",CV_WINDOW_AUTOSIZE);
294   ros::init(argc,argv,"red_localiser");
295   ImageConverter ic;
296   ros::spin();
297   return EXIT_SUCCESS;
298  }
```

Appendix G

C++ Code for FIFO Task Handler

```
#include <ros/ros.h>
#include <ploughing_reversible/task.h>
#include <ploughing_reversible/ReachPoint.h>
#include <ploughing_reversible/occupied.h>
#include <string>
#include <std_msgs/Bool.h>

#define hX 14.0000
#define hY 9.0000


using namespace std;

struct pose
{
  float x;
  float y;
};

//Method Signitures
list<pose> furrowLocalizer(pose start, pose end, float dist);
pose getComponent(list<pose> _list, int _i);
int getIndex(list<pose> _list, pose _pose);
//Variables
pose alpha,A,B,C,D, lastKnownPose;
list<pose> furCoordAB, furCoordCD, homeCD, homeAB;
float furDist, furTransAB, furTransCD;
string taskCmd="", pointName="";
int teamSize=0, rank=1, furTotal;
bool taskIsCompleted = false,
     targetIsReached = false,
     iAmConfident = false,
     furrowTransiting = false,
     pathIsCleared = false;
ploughing_reversible::ReachPoint reach;
ploughing_reversible::occupied checkPoint,updatePoint;
std_msgs::Bool imageRes;
class TaskHandler
{
  public:
    TaskHandler();
  private:
    ros::NodeHandle nh;
    ros::Subscriber task_cmd_sub;
    ros::Subscriber reach_point_status_sub;
    ros::Subscriber field_sub;
```

```
47      ros::Publisher reach_point_pub;
48      ros::ServiceClient ask_reference;
49      ros::ServiceClient update_reference;
50
51      void taskCmdCallback(const
52      ploughing_reversible::task::ConstPtr &msg);
53
54      void reachPointStatusCallback
55      (const ploughing_reversible::ReachPoint::ConstPtr &status);
56      void fieldCallback(const std_msgs::Bool::ConstPtr &imgStatus);
57  };
58  //Class Constructor
59  TaskHandler::TaskHandler()
60  {
61    task_cmd_sub = nh.subscribe<ploughing_reversible::task>
62    ("task_cmd",5,&TaskHandler::taskCmdCallback,this);
63
64    field_sub = nh.subscribe<std_msgs::Bool>
65    ("field_status",10,&TaskHandler::fieldCallback,this);
66
67    reach_point_status_sub = nh.subscribe<ploughing_reversible::
68    ReachPoint>("p1/reach_point_status",100,
69    &TaskHandler::reachPointStatusCallback,this);
70
71    reach_point_pub = nh.advertise<ploughing_reversible::
72    ReachPoint>("p1/requested_point",10);
73
74    ask_reference = nh.serviceClient<ploughing_reversible::
75    occupied>("check_coordinate");
76
77    update_reference = nh.serviceClient<ploughing_reversible::
78    occupied> ("add_coordinate");
79
80  }
81
82
83
84  //a method to receive the staus of reach point.
85  void TaskHandler::reachPointStatusCallback
86  (const ploughing_reversible::ReachPoint::ConstPtr &status)
87  {
88    pointName = status->hint;
89    pose newPose;
90    //case of reaching alpha
91    if (pointName.compare("alpha") == 0)
92    {
93      //go to first coordinate in AB set.
94      newPose = getComponent(furCoordAB,0);
95      reach.point.x = newPose.x;
96      reach.point.y = newPose.y;
97      reach.hint = "ab";
98      reach.status = false;
99      reach_point_pub.publish(reach);
100     iAmConfident = false;
101   }
102   //if my rank is known
103   if (iAmConfident)
104   {
105     pose reachedPoint;
106     reachedPoint.x = status->point.x;
```

```
107       reachedPoint.y = status->point.y;
108       /*
109         furrow transitioning:
110         stage1-> go back to the last position taken by the first rank robot.
111         stage2-> go to same x but y + furrowDistance.
112         stage3-> go to next target x but next target y + furrowDistance
113         stage4-> go to your starting point
114         resume your ploughing.
115       */
116       if (status->hint.compare("cd") == 0)
117       {
118         lastKnownPose.x = status->point.x;
119         lastKnownPose.y = status->point.y;
120         //continue to furrow transitioning
121         if (getIndex(furCoordCD,lastKnownPose)+teamSize <= furTotal-1)
122         {
123           newPose = getComponent(furCoordCD,
124           getIndex(furCoordCD,reachedPoint)-rank+1);
125
126           reach.point.x = newPose.x;
127           reach.point.y = newPose.y;
128           reach.hint = "stage1_cd";
129           reach.status = false;
130           reach_point_pub.publish(reach);
131
132         }
133         //go back home
134         if (getIndex(furCoordCD,lastKnownPose)+teamSize > furTotal-1)
135         {
136           newPose = getComponent(furCoordCD,
137           getIndex(furCoordCD,reachedPoint)-rank+1);
138
139           reach.point.x = newPose.x;
140           reach.point.y = newPose.y;
141           reach.hint = "home_cd_init";
142           reach.status = false;
143           reach_point_pub.publish(reach);
144         }
145       }
146       if (status->hint.compare("home_cd_init") == 0)
147         {
148           reach.point.x = reachedPoint.x;
149           reach.point.y = reachedPoint.y + furTransCD;
150           reach.hint = "home_cd";
151           reach.status = false;
152           reach_point_pub.publish(reach);
153         }
154       if (status->hint.compare("stage1_cd") == 0)
155       {
156         reach.point.x = reachedPoint.x;
157         reach.point.y = reachedPoint.y+furTransCD;
158         reach.hint = "stage2_cd";
159         reach.status = false;
160         reach_point_pub.publish(reach);
161       }
162       if (status->hint.compare("stage2_cd") == 0)
163       {
164         newPose = getComponent(furCoordCD,
165         getIndex(furCoordCD,lastKnownPose)+teamSize);
166
```

```
167         reach.point.x = newPose.x;
168         reach.point.y = newPose.y+furTransCD;
169         reach.hint = "stage3_cd";
170         reach.status = false;
171         reach_point_pub.publish(reach);
172       }
173       if (status->hint.compare("stage3_cd") == 0)
174       {
175         /*If the you are the first in rank,
176         then wait for the path to cleared*/
177         if (rank == 1 && teamSize > 1)
178         {
179           if(!pathIsCleared)
180           {
181             newPose = getComponent(furCoordCD,
182             getIndex(furCoordCD,lastKnownPose)+teamSize);
183
184             reach.point.x = newPose.x;
185             reach.point.y = newPose.y+furTransCD;
186             reach.hint = "stage3_cd";
187             reach.status = false;
188             reach_point_pub.publish(reach);
189           }
190           if (pathIsCleared)
191           {
192             newPose = getComponent(furCoordCD,
193             getIndex(furCoordCD,lastKnownPose)+teamSize);
194
195             reach.point.x = newPose.x;
196             reach.point.y = newPose.y;
197             reach.hint = "stage4_cd";
198             reach.status = false;
199             reach_point_pub.publish(reach);
200           }
201         }
202         if (rank != 1 || teamSize == 1)
203         {
204           newPose = getComponent(furCoordCD,
205           getIndex(furCoordCD,lastKnownPose)+teamSize);
206
207           reach.point.x = newPose.x;
208           reach.point.y = newPose.y;
209           reach.hint = "stage4_cd";
210           reach.status = false;
211           reach_point_pub.publish(reach);
212         }
213       }
214       if (status->hint.compare("stage4_cd") == 0)
215       {
216         pathIsCleared = false;
217         newPose = getComponent(furCoordAB,
218         getIndex(furCoordCD,lastKnownPose)+teamSize);
219
220         reach.point.x = newPose.x;
221         reach.point.y = newPose.y;
222         reach.hint = "ab";
223         reach.status = false;
224         reach_point_pub.publish(reach);
225       }
226       else if (status->hint.compare("ab") == 0)
```

```
227     {
228       lastKnownPose.x = status->point.x;
229       lastKnownPose.y = status->point.y;
230       //continue to furrow transitioning
231       if (getIndex(furCoordAB,lastKnownPose)+teamSize <= furTotal-1)
232       {
233         newPose = getComponent(furCoordAB,
234         getIndex(furCoordAB,reachedPoint)-rank+1);
235 
236         reach.point.x = newPose.x;
237         reach.point.y = newPose.y;
238         reach.hint = "stage1_ab";
239         reach.status = false;
240         reach_point_pub.publish(reach);
241       }
242       //go back home
243       if (getIndex(furCoordAB,lastKnownPose)+teamSize > furTotal-1)
244       {
245         newPose = getComponent(furCoordAB,
246         getIndex(furCoordAB,reachedPoint)-rank+1);
247 
248         reach.point.x = newPose.x;
249         reach.point.y = newPose.y;
250         reach.hint = "home_ab_init";
251         reach.status = false;
252         reach_point_pub.publish(reach);
253       }
254     }
255     if (status->hint.compare("home_ab_init") == 0)
256     {
257       reach.point.x = reachedPoint.x;
258       reach.point.y = reachedPoint.y-furTransAB;
259       reach.hint = "home_ab";
260       reach.status = false;
261       reach_point_pub.publish(reach);
262     }
263     if (status->hint.compare("stage1_ab") == 0)
264     {
265       reach.point.x = reachedPoint.x;
266       reach.point.y = reachedPoint.y-furTransAB;
267       reach.hint = "stage2_ab";
268       reach.status = false;
269       reach_point_pub.publish(reach);
270     }
271     if (status->hint.compare("stage2_ab") == 0)
272     {
273       newPose = getComponent(furCoordAB,
274       getIndex(furCoordAB,lastKnownPose)+teamSize);
275 
276       reach.point.x = newPose.x;
277       reach.point.y = newPose.y-furTransAB;
278       reach.hint = "stage3_ab";
279       reach.status = false;
280       reach_point_pub.publish(reach);
281     }
282     if (status->hint.compare("stage3_ab") == 0)
283     {
284       /*If the you are the first in rank, then wait for the path to cleared*/
285       if (rank == 1 && teamSize > 1)
286       {
```

```
287           if(!pathIsCleared)
288           {
289             newPose = getComponent(furCoordAB,
290             getIndex(furCoordAB,lastKnownPose)+teamSize);
291
292             reach.point.x = newPose.x;
293             reach.point.y = newPose.y-furTransAB;
294             reach.hint = "stage3_ab";
295             reach.status = false;
296             reach_point_pub.publish(reach);
297           }
298           if (pathIsCleared)
299           {
300             newPose = getComponent(furCoordAB,
301             getIndex(furCoordAB,lastKnownPose)+teamSize);
302
303             reach.point.x = newPose.x;
304             reach.point.y = newPose.y;
305             reach.hint = "stage4_ab";
306             reach.status = false;
307             reach_point_pub.publish(reach);
308           }
309         }
310         if (rank != 1 || teamSize == 1)
311         {
312           newPose = getComponent(furCoordAB,
313           getIndex(furCoordAB,lastKnownPose)+teamSize);
314
315           reach.point.x = newPose.x;
316           reach.point.y = newPose.y;
317           reach.hint = "stage4_ab";
318           reach.status = false;
319           reach_point_pub.publish(reach);
320         }
321       }
322       if (status->hint.compare("stage4_ab") == 0)
323       {
324         pathIsCleared = false;
325         newPose = getComponent(furCoordCD,
326         getIndex(furCoordAB,lastKnownPose)+teamSize);
327
328         reach.point.x = newPose.x;
329         reach.point.y = newPose.y;
330         reach.hint = "cd";
331         reach.status = false;
332         reach_point_pub.publish(reach);
333       }
334     }
335     //If the robot is not sure what rank it has
336     if (!iAmConfident && !furrowTransiting)
337     {
338       if (pointName.compare("ab") == 0)
339       {
340         pose temp = getComponent(furCoordAB,rank-1);
341         checkPoint.request.x = temp.x;
342         checkPoint.request.y = temp.y;
343         if(ask_reference.call(checkPoint))
344         {
345           //if the requested point is occupied
346           if (checkPoint.response.status == true)
```

```
347           {
348             //update your rank
349             rank++;
350             //go to the next ploughing target
351             newPose = getComponent(furCoordAB,rank-1);
352             reach.point.x = newPose.x;
353             reach.point.y = newPose.y;
354             reach.hint = "ab";
355             reach.status = true;
356             reach_point_pub.publish(reach);
357           }
358           //if the requested point is not occupied
359           else if (checkPoint.response.status == false)
360           {
361             //update reference
362             pose temp = getComponent(furCoordAB,rank-1);
363             updatePoint.request.x = temp.x;
364             updatePoint.request.y = temp.y;
365             if(update_reference.call(updatePoint))
366               {
367               //start plouging
368               newPose = getComponent(furCoordCD,rank-1);
369               reach.point.x = newPose.x;
370               reach.point.y = newPose.y;
371               reach.hint = "cd";
372               reach.status = false;
373               reach_point_pub.publish(reach);
374               iAmConfident = true;
375               furrowTransiting = true;
376             }
377           }
378         }
379       }
380     }
381     //Going back to home position through cd sideline
382     if (status->hint.compare("home_cd") == 0)
383     {
384       pose reachedPoint;
385       reachedPoint.x = status->point.x;
386       reachedPoint.y = status->point.y;
387       if (getIndex(homeCD,reachedPoint)-1 > -1)
388       {
389         newPose = getComponent(homeCD,
390         getIndex(homeCD,reachedPoint)-1);
391
392         reach.point.x = newPose.x;
393         reach.point.y = newPose.y;
394         reach.hint = "home_cd";
395         reach.status = false;
396         reach_point_pub.publish(reach);
397       }
398       if (getIndex(homeCD,reachedPoint)-1 == -1)
399       {
400         reach.point.x = hX;
401         reach.point.y = hY;
402         reach.hint = "home";
403         reach.status = false;
404         reach_point_pub.publish(reach);
405       }
406     }
```

```
407     //Going back to home position through ab sideline
408     if (status->hint.compare("home_ab") == 0)
409     {
410       pose reachedPoint;
411       reachedPoint.x = status->point.x;
412       reachedPoint.y = status->point.y;
413       if (getIndex(homeAB,reachedPoint)-1 > -1)
414       {
415         newPose = getComponent(homeAB,
416         getIndex(homeAB,reachedPoint)-1);
417 
418         reach.point.x = newPose.x;
419         reach.point.y = newPose.y;
420         reach.hint = "home_ab";
421         reach.status = false;
422         reach_point_pub.publish(reach);
423       }
424       if (getIndex(homeAB,reachedPoint)-1 == -1)
425       {
426         reach.point.x = hX;
427         reach.point.y = hY;
428         reach.hint = "home";
429         reach.status = false;
430         reach_point_pub.publish(reach);
431       }
432     }
433   }
434   //a method to receive status of the field
435   void TaskHandler::fieldCallback(const
436   std_msgs::Bool::ConstPtr &fieldStatus)
437   {
438     //to avoid unwanted reading during
439     //other stages of the process
440     if (pointName.compare("stage3_ab") == 0
441     || pointName.compare("stage3_cd") == 0)
442     {
443       pathIsCleared = fieldStatus->data;
444       ROS_INFO("Image Response Received");
445     }
446   }
447   //a method to receive the task command
448   void TaskHandler::taskCmdCallback(const
449   ploughing_reversible::task::ConstPtr &msg)
450   {
451     //loading data to appropriate variables
452     taskCmd = msg->task;
453     alpha.x = msg->alpha.x;
454     alpha.y = msg->alpha.y;
455     A.x = msg->a.x;
456     A.y = msg->a.y;
457     B.x = msg->b.x;
458     B.y = msg->b.y;
459     C.x = msg->c.x;
460     C.y = msg->c.y;
461     D.x = msg->d.x;
462     D.y = msg->d.y;
463     furDist = msg->furrow_distance;
464     furTransAB = msg->transition_area_ab;
465     furTransCD = msg->transition_area_cd;
466     teamSize = msg->team_size;
```

```
467    // calculate furrow coordinates
468    furCoordAB = furrowLocalizer(A,B,furDist);
469    furCoordCD = furrowLocalizer(C,D,furDist);
470    furTotal = furCoordCD.size();
471    //calculating homing coordinates
472    C.y = C.y + furTransCD;
473    D.y = D.y + furTransCD;
474    A.y = A.y - furTransAB;
475    B.y = B.y - furTransAB;
476    homeCD = furrowLocalizer(C,D,furDist);
477    homeAB = furrowLocalizer(A,B,furDist);
478
479    /*the first step is to go to alpha*/
480    reach.point.x = alpha.x;
481    reach.point.y = alpha.y;
482    reach.hint = "alpha";
483    reach.status = false;
484    reach_point_pub.publish(reach);
485
486  }
487
488  //main method
489  int main(int argc, char** argv)
490  {
491    ros::init(argc,argv,"r1_task_handler");
492    TaskHandler th;
493    ros::spin();
494    return 0;
495  }
496  //a method to map the furrow coordinates.
497  //it collects all calculated coordinates in a list.
498  list<pose> furrowLocalizer(pose start, pose end, float dist)
499  {
500    list<pose> createdList;
501    pose currentPose = start;
502    while(currentPose.x >= end.x)
503    {
504      /*
505        push_back -> add element at the end, last element.
506        push_front -> add element to the front, element 0.
507      */
508      createdList.push_back(currentPose);
509      currentPose.x -= dist;
510    }
511    return createdList;
512  }
513  //a method to get particular component in a given list.
514  pose getComponent(list<pose> _list, int _i){
515      list<pose>::iterator it = _list.begin();
516      for(int i=0; i<_i; i++){
517          ++it;
518      }
519      return *it;
520  }
521  int getIndex(list<pose> _list, pose _pose)
522  {
523    int index = -1;
524    pose temp;
525    for(int i=0; i<_list.size(); i++)
526    {
```

```
527             temp = getComponent(_list,i);
528             if (temp.x == _pose.x && temp.y == _pose.y)
529             {
530               index = i;
531             }
532     }
533     return index;
534 }
```

Appendix H

C++ Code for LIFO Task Handler

```
#include <ros/ros.h>
#include <ploughing_reversible/task.h>
#include <ploughing_reversible/ReachPoint.h>
#include <ploughing_reversible/occupied.h>
#include <string>
#include <std_msgs/Bool.h>
#include <stdio.h>
#include <iostream>

#define hX 14.0000
#define hY 9.0000
#define tranX 0.8
#define tranY 1


using namespace std;

struct pose
{
  float x;
  float y;
};

//Method Signitures
list<pose> furrowLocalizer(pose start, pose end, float dist);
pose getComponent(list<pose> _list, int _i);
int getIndex(list<pose> _list, pose _pose);
//Variables
pose alpha,A,B,C,D, lastKnownPose, nextPose;
list<pose> furCoordAB, furCoordCD, homeCD, homeAB;
float furDist, furTransAB, furTransCD;
string taskCmd="", pointName="";
int teamSize=0, rank=1, furTotal,
    ploughingTurn = 0,
    ploughingRound = 0;
bool taskIsCompleted = false,
     targetIsReached = false,
     iAmConfident = false,
     furrowTransiting = false,
     pathIsCleared = false;
ploughing_reversible::ReachPoint reach;
ploughing_reversible::occupied checkPoint,updatePoint;
std_msgs::Bool imageRes;
class TaskHandler
{
  public:
```

```
47      TaskHandler();
48    private:
49      ros::NodeHandle nh;
50      ros::Subscriber task_cmd_sub;
51      ros::Subscriber reach_point_status_sub;
52
53      ros::Publisher reach_point_pub;
54      ros::ServiceClient ask_reference;
55      ros::ServiceClient update_reference;
56
57      void taskCmdCallback(const
58      ploughing_reversible::task::ConstPtr &msg);
59
60      void reachPointStatusCallback(const
61      ploughing_reversible::ReachPoint::ConstPtr &status);
62
63      void fieldCallback(const
64      std_msgs::Bool::ConstPtr &imgStatus);
65  };
66  //Class Constructor
67  TaskHandler::TaskHandler()
68  {
69    task_cmd_sub = nh.subscribe<ploughing_reversible::task>
70    ("task_cmd",5,&TaskHandler::taskCmdCallback,this);
71
72    reach_point_status_sub = nh.subscribe<ploughing_reversible::
73    ReachPoint> ("p1/reach_point_status",100,&TaskHandler::
74    reachPointStatusCallback,this);
75
76    reach_point_pub = nh.advertise<ploughing_reversible::
77    ReachPoint>("p1/requested_point",10);
78
79    ask_reference = nh.serviceClient<ploughing_reversible::
80    occupied>("check_coordinate");
81
82    update_reference = nh.serviceClient<ploughing_reversible::
83    occupied>("add_coordinate");
84  }
85
86  //a method to receive the staus of reach point.
87  void TaskHandler::reachPointStatusCallback(const
88  ploughing_reversible::ReachPoint::ConstPtr &status)
89  {
90    pointName = status->hint;
91    pose newPose;
92    //Step 2-1: Enter the field by going
93    //to the coordinate of the first furrow
94    if (pointName.compare("alpha") == 0)
95    {
96      //go to first coordinate in AB set.
97      newPose = getComponent(furCoordAB,0);
98      reach.point.x = newPose.x;
99      reach.point.y = newPose.y;
100     reach.hint = "ab";
101     reach.status = false;
102     reach_point_pub.publish(reach);
103     iAmConfident = false;
104   }
105   if (iAmConfident)
106   {
```

```
107     //It is important to keep track of the current
108     // position of the robot when it is confident
109     pose reachedPoint;
110     reachedPoint.x = status->point.x;
111     reachedPoint.y = status->point.y;
112     //Next stage depends on the reached position of the robot (CD OR AB)
113     if (status->hint.compare("cd") == 0)
114     {
115       //Now this location will be saved as reference location
116       lastKnownPose.x = status->point.x;
117       lastKnownPose.y = status->point.y;
118      /*
119         Depending on round number different
120         equations will be used to determine next furrow.
121         Round number is either even or odd.
122      */
123         //check if the next round is available
124         if (getIndex(furCoordCD,lastKnownPose)+
125         2*(teamSize - rank)+1 <= furTotal-1)
126         {
127            nextPose = getComponent(furCoordCD,
128            getIndex(furCoordCD,lastKnownPose)+2*(teamSize - rank)+1);
129
130           if (rank != teamSize)
131           {
132             reach.point.x = reachedPoint.x - tranX;
133             reach.point.y = reachedPoint.y + tranY;
134             reach.hint = "stage1_cd";
135             reach.status = false;
136             reach_point_pub.publish(reach);
137           }
138           else if (rank == teamSize)
139           {
140
141             reach.point.x = nextPose.x;
142             reach.point.y = nextPose.y;
143             reach.hint = "stage3_cd";
144             reach.status = false;
145             reach_point_pub.publish(reach);
146           }
147         }
148         else if (getIndex(furCoordCD,lastKnownPose)
149         +2*(teamSize - rank)+1 > furTotal-1)
150         {
151           reach.point.x = reachedPoint.x;
152           reach.point.y = reachedPoint.y + tranY;
153           reach.hint = "home_cd";
154           reach.status = false;
155           reach_point_pub.publish(reach);
156         }
157     }
158     if (status->hint.compare("ab") == 0)
159     {
160       //Now this location will be saved as reference location
161       lastKnownPose.x = status->point.x;
162       lastKnownPose.y = status->point.y;
163      /*
164         Depending on round number different equations
165         will be used to determine next furrow.
166         Round number is either even or odd.
```

```
167       */
168         //when round is even (because the task is start from ab,
169         //when the robot reaches ab ploughingRound is always even)
170         //check if the next round is available
171         if (getIndex(furCoordAB,lastKnownPose)+2*rank-1 <= furTotal-1)
172         {
173
174           //log the next starting point before start ploughing
175           nextPose = getComponent(furCoordAB,
176           getIndex(furCoordAB,lastKnownPose)+2*rank-1);
177           //If robot rank is not the first in the team,
178           //then the robot is compelled to perform manuevering
179           if (rank != 1)
180           {
181               reach.point.x = reachedPoint.x - tranX;
182               reach.point.y = reachedPoint.y - tranY;
183               reach.hint = "stage1_ab";
184               reach.status = false;
185               reach_point_pub.publish(reach);
186           }
187           else if (rank == 1)
188           {
189
190               reach.point.x = nextPose.x;
191               reach.point.y = nextPose.y;
192               reach.hint = "stage3_ab";
193               reach.status = false;
194               reach_point_pub.publish(reach);
195           }
196         }
197         else if (getIndex(furCoordAB,lastKnownPose)+2*rank-1 > furTotal-1)
198         {
199           //starting going home
200           reach.point.x = reachedPoint.x;
201           reach.point.y = reachedPoint.y - tranY;
202           reach.hint = "home_ab";
203           reach.status = false;
204           reach_point_pub.publish(reach);
205         }
206       }
207       //execute the second stage of furrow transitioning on CD side
208       else if (status->hint.compare("stage1_cd") == 0)
209       {
210         reach.point.x = reachedPoint.x + 2*tranX;
211         reach.point.y = reachedPoint.y;
212         reach.hint = "stage2_cd";
213         reach.status = false;
214         reach_point_pub.publish(reach);
215       }
216       else if (status->hint.compare("stage1_ab") == 0)
217       {
218         reach.point.x = reachedPoint.x + 2*tranX;
219         reach.point.y = reachedPoint.y;
220         reach.hint = "stage2_ab";
221         reach.status = false;
222         reach_point_pub.publish(reach);
223       }
224       //wait until a message is received from
225       //horizon detector module is received
226       else if (status->hint.compare("go_cd") == 0)
```

```
227     {
228       //log the next starting point before start ploughing
229       nextPose = getComponent(furCoordCD,
230       getIndex(furCoordCD,lastKnownPose)+2*(teamSize - rank)+1);
231
232       //nextPose logged from stage1_cd.
233       reach.point.x = nextPose.x;
234       reach.point.y = nextPose.y;
235       reach.hint = "stage3_cd";
236       reach.status = false;
237       reach_point_pub.publish(reach);
238     }
239     else if (status->hint.compare("go_ab") == 0)
240     {
241       nextPose = getComponent(furCoordAB,
242       getIndex(furCoordAB,lastKnownPose)+2*rank-1);
243       //nextPose logged from stage1_ab.
244       reach.point.x = nextPose.x;
245       reach.point.y = nextPose.y;
246       reach.hint = "stage3_ab";
247       reach.status = false;
248       reach_point_pub.publish(reach);
249     }
250     //when you reached the starting point,
251     //find your next target in AB set and plough.
252     //the target in AB set is mirror of the reached CD.
253     else if (status->hint.compare("stage3_cd") == 0)
254     {
255       nextPose = getComponent(furCoordAB,
256       getIndex(furCoordCD,reachedPoint));
257
258       reach.point.x = nextPose.x;
259       reach.point.y = nextPose.y;
260       reach.hint = "ab";
261       reach.status = false;
262       reach_point_pub.publish(reach);
263       ploughingRound++;
264     }
265     else if (status->hint.compare("stage3_ab") == 0)
266     {
267       nextPose = getComponent(furCoordCD,
268       getIndex(furCoordAB,reachedPoint));
269
270       reach.point.x = nextPose.x;
271       reach.point.y = nextPose.y;
272       reach.hint = "cd";
273       reach.status = false;
274       reach_point_pub.publish(reach);
275       ploughingRound++;
276     }
277     //Going back to home position through cd sideline
278         if (status->hint.compare("home_cd") == 0)
279         {
280             pose reachedPoint;
281             reachedPoint.x = status->point.x;
282             reachedPoint.y = status->point.y;
283             if (getIndex(homeCD,reachedPoint)-1 > -1)
284             {
285                 newPose = getComponent(homeCD,
286                 getIndex(homeCD,reachedPoint)-1);
```

```
287
288                 reach.point.x = newPose.x;
289                 reach.point.y = newPose.y;
290                 reach.hint = "home_cd";
291                 reach.status = false;
292                 reach_point_pub.publish(reach);
293             }
294             if (getIndex(homeCD,reachedPoint)-1 == -1)
295             {
296                 reach.point.x = hX;
297                 reach.point.y = hY;
298                 reach.hint = "home";
299                 reach.status = false;
300                 reach_point_pub.publish(reach);
301             }
302         }
303         //Going back to home position through ab sideline
304         if (status->hint.compare("home_ab") == 0)
305         {
306             pose reachedPoint;
307             reachedPoint.x = status->point.x;
308             reachedPoint.y = status->point.y;
309             if (getIndex(homeAB,reachedPoint)-1 > -1)
310             {
311                 newPose = getComponent(homeAB,
312                 getIndex(homeAB,reachedPoint)-1);
313
314                 reach.point.x = newPose.x;
315                 reach.point.y = newPose.y;
316                 reach.hint = "home_ab";
317                 reach.status = false;
318                 reach_point_pub.publish(reach);
319             }
320             if (getIndex(homeAB,reachedPoint)-1 == -1)
321             {
322                 reach.point.x = hX;
323                 reach.point.y = hY;
324                 reach.hint = "home";
325                 reach.status = false;
326                 reach_point_pub.publish(reach);
327             }
328         }
329     }
330   //robot will be confident whenever it finds an unploughed furrow
331   else if (!iAmConfident)
332   {
333     if (pointName.compare("ab") == 0)
334     {
335       pose temp = getComponent(furCoordAB,rank-1);
336       checkPoint.request.x = temp.x;
337       checkPoint.request.y = temp.y;
338       if(ask_reference.call(checkPoint))
339       {
340         //if the requested point is occupied
341         if (checkPoint.response.status == true)
342         {
343           //update your rank
344           rank++;
345           //go to the next ploughing target
346           newPose = getComponent(furCoordAB,rank-1);
```

```
347             reach.point.x = newPose.x;
348             reach.point.y = newPose.y;
349             reach.hint = "ab";
350             reach.status = true;
351             reach_point_pub.publish(reach);
352           }
353           //if the requested point is not occupied
354           else if (checkPoint.response.status == false)
355           {
356             //update reference database for the next robot
357             pose temp = getComponent(furCoordAB,rank-1);
358             updatePoint.request.x = temp.x;
359             updatePoint.request.y = temp.y;
360             if(update_reference.call(updatePoint))
361             {
362               //start ploughing
363               newPose = getComponent(furCoordCD,rank-1);
364               reach.point.x = newPose.x;
365               reach.point.y = newPose.y;
366               reach.hint = "cd";
367               reach.status = false;
368               reach_point_pub.publish(reach);
369               iAmConfident = true;
370               ploughingRound++;
371             }
372           }
373         }
374       }
375     }
376   }
377
378   //a method to receive the task command
379   void TaskHandler::taskCmdCallback(const
380   ploughing_reversible::task::ConstPtr &msg)
381   {
382     //loading data to appropriate variables
383     taskCmd = msg->task;
384     alpha.x = msg->alpha.x;
385     alpha.y = msg->alpha.y;
386     A.x = msg->a.x;
387     A.y = msg->a.y;
388     B.x = msg->b.x;
389     B.y = msg->b.y;
390     C.x = msg->c.x;
391     C.y = msg->c.y;
392     D.x = msg->d.x;
393     D.y = msg->d.y;
394     furDist = msg->furrow_distance;
395     teamSize = msg->team_size;
396     // calculate furrow coordinates
397     furCoordAB = furrowLocalizer(A,B,furDist);
398     furCoordCD = furrowLocalizer(C,D,furDist);
399     furTotal = furCoordCD.size();
400     //calculating homing coordinates
401     C.y = C.y + tranY;
402     D.y = D.y + tranY;
403     A.y = A.y - tranY;
404     B.y = B.y - tranY;
405     homeCD = furrowLocalizer(C,D,furDist);
406     homeAB = furrowLocalizer(A,B,furDist);
```

```
407   //Step 1: Go to point alpha
408   reach.point.x = alpha.x;
409   reach.point.y = alpha.y;
410   reach.hint = "alpha";
411   reach.status = false;
412   reach_point_pub.publish(reach);
413 }
414
415 //main method
416 int main(int argc, char** argv)
417 {
418   ros::init(argc,argv,"pra1_th");
419   TaskHandler th;
420   ros::spin();
421   return 0;
422 }
423 //a method to map the furrow coordinates.
424 //it collects all calculated coordinates in a list.
425 list<pose> furrowLocalizer(pose start, pose end, float dist)
426 {
427   list<pose> createdList;
428   pose currentPose = start;
429   while(currentPose.x >= end.x)
430   {
431     /*
432       push_back -> add element at the end, last element.
433       push_front -> add element to the front, element 0.
434     */
435     createdList.push_back(currentPose);
436     currentPose.x -= dist;
437   }
438   return createdList;
439 }
440 //a method to get particular component in a given list.
441 pose getComponent(list<pose> _list, int _i){
442     list<pose>::iterator it = _list.begin();
443     for(int i=0; i<_i; i++){
444         ++it;
445     }
446     return *it;
447 }
448 int getIndex(list<pose> _list, pose _pose)
449 {
450   int index = -1;
451   pose temp;
452   for(int i=0; i<_list.size(); i++)
453   {
454         temp = getComponent(_list,i);
455         if (temp.x == _pose.x && temp.y == _pose.y)
456         {
457           index = i;
458         }
459   }
460   return index;
461 }
```

Appendix I

C++ Code for Self-organised Task Handler

```
#include <ros/ros.h>
#include <iostream>
#include <sstream>
#include <nav_msgs/Odometry.h>
#include <ploughing_reversible/ReachPoint.h>
#include <ploughing_reversible/occupied.h>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */

using namespace std;

int robot_id = 0;
string robot_name;
float cpX, cpY;
bool sysIsReady = false;
int p_index = 0;
struct pose
{
  float x;
  float y;
};

//Method Signitures
list<pose> furrowLocalizer(pose start, pose end, float dist);
pose getComponent(list<pose> _list, int _i);
int getIndex(list<pose> _list, pose _pose);
void map_init(void);
//Variables
pose alpha,A,B,C,D, lastKnownPose, newPose, topAd, bottomAd, leftAd;
list<pose> furCoordAB, furCoordCD, homeCD, homeAB;
float furDist, furTransAB, furTransCD;
int furTotal;
ploughing_reversible::ReachPoint reach;
ploughing_reversible::occupied checkPoint,updatePoint;

ros::Publisher reach_point_pub;


class THClass
{
  public:
    THClass();
  private:
    //a normal node handle for general tasks
    ros::NodeHandle n;
```

```
47      ros::Subscriber pose_sub;
48      ros::Subscriber reach_point_status_sub;
49
50      ros::ServiceClient ask_reference;
51      ros::ServiceClient update_reference;
52
53
54      void reachPointStatusCallback(const ploughing_reversible
55           ::ReachPoint::ConstPtr &reached_point);
56      void poseCB(const nav_msgs::Odometry::ConstPtr &pose);
57  };
58
59  THClass::THClass()
60  {
61    //a private node handle for collecting parameters
62    ros::NodeHandle nh("~");
63    ostringstream ss;
64    nh.getParam("robot_id",robot_id);
65    ss << "robot_";
66    ss << robot_id;
67    robot_name = ss.str();
68    string pose_pipename = robot_name+"/base_pose_ground_truth";
69    string pointreq_pipename = robot_name + "/requested_point";
70    string pointres_pointname = robot_name + "/reach_point_status";
71    pose_sub = n.subscribe<nav_msgs::Odometry>
72                (pose_pipename,10,&THClass::poseCB,this);
73    reach_point_status_sub = n.subscribe<ploughing_reversible::ReachPoint>
74          (pointres_pointname,100,&THClass::reachPointStatusCallback,this);
75    reach_point_pub = n.advertise<ploughing_reversible::ReachPoint>
76                      (pointreq_pipename,10);
77    ask_reference = n.serviceClient<ploughing_reversible::occupied>
78                      ("check_coordinate");
79    update_reference = n.serviceClient<ploughing_reversible::occupied>
80                      ("add_coordinate");
81  }
82  void THClass::poseCB(const nav_msgs::Odometry::ConstPtr &pose)
83  {
84    cpX = pose->pose.pose.position.x;
85    cpY = pose->pose.pose.position.y;
86
87    if (sysIsReady && reach_point_pub.getNumSubscribers() != 0)
88    {
89      if (cpX <= C.x && cpY >= C.y)
90      {
91        reach.point.x = topAd.x;
92        reach.point.y = topAd.y;
93        reach.hint = "ab1-adjust";
94        reach.status = false;
95        reach_point_pub.publish(reach);
96      }
97      else if (cpX <= B.x && cpY >= B.y && cpY <= D.y)
98      {
99        reach.point.x = bottomAd.x;
100       reach.point.y = bottomAd.y;
101       reach.hint = "ab1-adjust";
102       reach.status = false;
103       reach_point_pub.publish(reach);
104     }
105     else
106     {
```

```
107         newPose = getComponent(furCoordAB,0);
108         reach.point.x = newPose.x;
109         reach.point.y = newPose.y;
110         reach.hint = "ab1";
111         reach.status = false;
112         reach_point_pub.publish(reach);
113       }
114       sysIsReady = false;
115     }
116   }
117
118   void THClass::reachPointStatusCallback
119       (const ploughing_reversible::ReachPoint::ConstPtr &reached_point)
120   {
121     string target_name = reached_point->hint;
122     pose reached_location;
123     reached_location.x = reached_point->point.x;
124     reached_location.y = reached_point->point.y;
125     if (target_name.compare("ab1-left") == 0)
126     {
127       reach.point.x = bottomAd.x;
128       reach.point.y = bottomAd.y;
129       reach.hint = "ab1-adjust";
130       reach.status = false;
131       reach_point_pub.publish(reach);
132     }
133     if (target_name.compare("ab1-adjust") == 0)
134     {
135       if (p_index < int(furTotal/2))
136       {
137         pose newPose = getComponent(furCoordAB,0);
138         reach.point.x = newPose.x;
139         reach.point.y = newPose.y;
140         reach.hint = "ab1";
141         reach.status = false;
142         reach_point_pub.publish(reach);
143       }
144       else if (p_index >= int(furTotal/2))
145       {
146         pose newPose = getComponent(furCoordAB,p_index);
147         reach.point.x = newPose.x;
148         reach.point.y = newPose.y;
149         reach.hint = "ab-resume";
150         reach.status = false;
151         reach_point_pub.publish(reach);
152       }
153     }
154     if (target_name.compare("ab1") == 0 ||
155        target_name.compare("ab") == 0 ||
156        target_name.compare("ab-resume") == 0)
157     {
158       pose temp = getComponent(furCoordAB,p_index);
159       checkPoint.request.x = temp.x;
160       checkPoint.request.y = temp.y;
161       if (p_index < furTotal - 1)
162       {
163         if(ask_reference.call(checkPoint))
164         {
165             //if the requested point is occupied
166             if (checkPoint.response.status == true)
```

```
167             {
168               //go to the next ploughing target
169               p_index = p_index + 1;
170               newPose = getComponent(furCoordAB,p_index);
171               reach.point.x = newPose.x;
172               reach.point.y = newPose.y;
173               reach.hint = "ab";
174               reach.status = true;
175               reach_point_pub.publish(reach);
176             }
177             //if the requested point is not occupied.
178             else if (checkPoint.response.status == false)
179             {
180               //update reference
181               pose temp = getComponent(furCoordAB,p_index);
182
183               updatePoint.request.x = temp.x;
184               updatePoint.request.y = temp.y;
185               if(update_reference.call(updatePoint))
186               {
187                   //start plouging
188                   newPose = getComponent(furCoordCD,p_index);
189                   reach.point.x = newPose.x;
190                   reach.point.y = newPose.y;
191                   reach.hint = "cd";
192                   reach.status = false;
193                   reach_point_pub.publish(reach);
194               }
195             }
196         }
197       }
198       else if (p_index >= furTotal-1)
199       {
200         srand (time(NULL));
201         float randx = -1*rand()%60+30; //a random number between 5 and 15
202         usleep(500000);
203         srand (time(NULL));
204         float randy = -1*(rand()%30+15); //a random number between -1 and 1
205
206
207         reach.point.x = cpX + randx;
208         reach.point.y = cpY + randy;
209         reach.hint = "done";
210         reach.status = false;
211         reach_point_pub.publish(reach);
212       }
213     }
214     if (target_name.compare("cd") == 0)
215     {
216       if (getIndex(furCoordCD,reached_location) < furTotal-1)
217       {
218         //decide which direction is closer
219         if (getIndex(furCoordCD,reached_location) >= int(furTotal/2))
220         {
221           reach.point.x = leftAd.x;
222           reach.point.y = leftAd.y;
223           reach.hint = "ab1-left";
224           reach.status = false;
225           reach_point_pub.publish(reach);
226         }
```

```
227         else if (getIndex(furCoordCD,reached_location) < int(furTotal/2))
228         {
229           reach.point.x = topAd.x;
230           reach.point.y = topAd.y;
231           reach.hint = "ab1-adjust";
232           reach.status = false;
233           reach_point_pub.publish(reach);
234         }
235       }
236       else
237       {
238         reach.point.x = cpX;
239         reach.point.y = cpY+3.0;
240         reach.hint = "done";
241         reach.status = false;
242         reach_point_pub.publish(reach);
243       }
244     }
245   }
246   int main(int argc, char **argv)
247   {
248     map_init();
249     ros::init(argc,argv,"task_handler");
250     THClass th;
251     sleep(3);
252     sysIsReady = true;
253
254     ros::spin();
255     return 0;
256   }
257   //a method to map the furrow coordinates.
258   //it collects all calculated coordinates in a list.
259   list<pose> furrowLocalizer(pose start, pose end, float dist)
260   {
261     list<pose> createdList;
262     pose currentPose = start;
263     while(currentPose.x >= end.x)
264     {
265       /*
266         push_back -> add element at the end, last element.
267         push_front -> add element to the front, element 0.
268       */
269       createdList.push_back(currentPose);
270       currentPose.x -= dist;
271     }
272     return createdList;
273   }
274   //a method to get particular component in a given list.
275   pose getComponent(list<pose> _list, int _i){
276       list<pose>::iterator it = _list.begin();
277       for(int i=0; i<_i; i++){
278           ++it;
279       }
280       return *it;
281   }
282   int getIndex(list<pose> _list, pose _pose)
283   {
284     int index = -1;
285     pose temp;
286     for(int i=0; i<_list.size(); i++)
```

```
287     {
288             temp = getComponent(_list,i);
289             if (temp.x == _pose.x && temp.y == _pose.y)
290             {
291               index = i;
292             }
293     }
294     return index;
295   }
296   void map_init(void)
297   {
298     A.x = 10.0;
299     A.y = -13.0;
300     B.x = -10.0;
301     B.y = -13.0;
302     C.x = 10.0;
303     C.y = 13.0;
304     D.x = -10.0;
305     D.y = 13.0;
306     bottomAd.x = B.x - 5.0;
307     bottomAd.y = B.y - 5.0;
308     topAd.x = C.x + 5.0;
309     topAd.y = C.y + 1.0;
310     leftAd.x = D.x - 6.0;
311     leftAd.y = D.y + 1.0;
312     furDist = 0.4;
313     furTransAB = 4.0;
314     furTransCD = 4.0;
315     // calculate furrow coordinates
316     furCoordAB = furrowLocalizer(A,B,furDist);
317     furCoordCD = furrowLocalizer(C,D,furDist);
318     furTotal = furCoordCD.size();
319     //calculating homing coordinates
320     C.y = C.y + furTransCD;
321     D.y = D.y + furTransCD;
322     A.y = A.y - furTransAB;
323     B.y = B.y - furTransAB;
324     homeCD = furrowLocalizer(C,D,furDist);
325     homeAB = furrowLocalizer(A,B,furDist);
326
327   }
```

Appendix J

C++ Code for Region-based Task Handler Using ROS

```
#include <ros/ros.h>
#include <ploughing_reversible/task.h>
#include <ploughing_reversible/ReachPoint.h>
#include <ploughing_reversible/occupied.h>
#include <nav_msgs/Odometry.h>
#include <string>
#include <std_msgs/Bool.h>
#include <stdio.h>
#include <iostream>
#include <math.h>

#define hX 14.0000
#define hY 9.0000
#define tranX 0.8
#define tranY 3.0
#define PI 3.14159265

using namespace std;

struct pose
{
    float x;
    float y;
};

//Method Signitures
list<pose> furrowLocalizer(pose start, pose end, float dist);
pose getComponent(list<pose> _list, int _i);
int getIndex(list<pose> _list, pose _pose);
//Variables
pose alpha,A,B,C,D,
     nextPose, beta,
     gama, psi, rho,
     currentPose;
list<pose> furCoordAB, furCoordCD, homeCD, homeAB;
float furDist, furTransAB, furTransCD;
string taskCmd="", pointName="";
int teamSize=0,
    rank=1,
    furTotal,
    ploughingTurn = 0,
    ploughingRound = 0;
bool taskIsCompleted = false,
     targetIsReached = false,
     iAmConfident = false,
     pathIsCleared = false,
```

```
47        targetIsOccupied = false;
48  ploughing_reversible::ReachPoint reach;
49  ploughing_reversible::occupied checkPoint,updatePoint;
50  std_msgs::Bool imageRes;
51  /*A vector based integer for regions with at least 3 memebers.
52  initially the values are set to zero*/
53  vector<int> region(3);
54  /*A vector based integer for starting furNum of each furrow*/
55  vector<int> furNum(3);
56  /*A vector based pose for start point of the each region.
57  It is only on AB side as robots all goes to here first.*/
58  vector<pose> regPoint(3);
59  //An integer to count the regions
60  int regionCounter = 0;
61  //An integer to count number of processed tracks
62  int numOfProcessedTracks = 0;
63  class TaskHandler
64  {
65  public:
66      TaskHandler();
67  private:
68      ros::NodeHandle nh;
69      ros::Subscriber task_cmd_sub;
70      ros::Subscriber reach_point_status_sub;
71      ros::Subscriber img_sub;
72      ros::Subscriber pose_sub;
73
74
75      ros::Publisher reach_point_pub;
76      ros::ServiceClient ask_reference;
77      ros::ServiceClient update_reference;
78
79      void taskCmdCallback
80      (const ploughing_reversible::task::ConstPtr &msg);
81      void reachPointStatusCallback
82      (const ploughing_reversible::ReachPoint::ConstPtr &status);
83      void imgCB(const std_msgs::Bool::ConstPtr &imgStatus);
84      void poseCB(const nav_msgs::Odometry::ConstPtr &pose);
85  };
86  //Class Constructor
87  TaskHandler::TaskHandler()
88  {
89      task_cmd_sub = nh.subscribe<ploughing_reversible::task>
90      ("task_cmd",5,&TaskHandler::taskCmdCallback,this);
91
92      reach_point_status_sub = nh.subscribe<ploughing_reversible::
93      ReachPoint>("p1/reach_point_status",100,&TaskHandler::
94      reachPointStatusCallback,this);
95
96      reach_point_pub = nh.advertise<ploughing_reversible::ReachPoint>
97      ("p1/requested_point",10);
98
99      ask_reference = nh.serviceClient<ploughing_reversible::occupied>
100     ("check_coordinate");
101     update_reference = nh.serviceClient<ploughing_reversible::occupied>
102     ("add_coordinate");
103
104     img_sub = nh.subscribe<std_msgs::Bool>
105     ("r1_img_response",10,&TaskHandler::imgCB,this);
106
```

```
107     pose_sub = nh.subscribe<nav_msgs::Odometry>
108     ("robot_0/base_pose_ground_truth",10,&TaskHandler::poseCB,this);
109 }
110
111 //a method to receive the staus of reach point.
112 void TaskHandler::reachPointStatusCallback
113     (const ploughing_reversible::ReachPoint::ConstPtr &status)
114 {
115     pointName = status->hint;
116     pose newPose;
117     //Step 2-1: go to the first region
118     if (pointName.compare("alpha") == 0)
119     {
120         //go to first coordinate in AB set for regions.
121         newPose = regPoint[regionCounter];
122         reach.point.x = newPose.x;
123         //keep your distance from start line until
124         //you are certain that no one is in the space.
125         reach.point.y = newPose.y-tranY;
126         reach.hint = "ab_reg";
127         reach.status = false;
128         reach_point_pub.publish(reach);
129         iAmConfident = false;
130     }
131     if (iAmConfident)
132     {
133         if (status->hint.compare("ab_start") == 0)
134         {
135             pose newPose = getComponent(furCoordCD,
136             getIndex(furCoordAB,regPoint[regionCounter]));
137
138             reach.point.x = newPose.x;
139             reach.point.y = newPose.y;
140             reach.hint = "cd";
141             reach.status = false;
142             reach_point_pub.publish(reach);
143             numOfProcessedTracks++;
144         }
145         if (status->hint.compare("cd") == 0)
146         {
147             //If number of processed tracks is smaller than
148             //number of tracks in your region then transit your track
149             if (numOfProcessedTracks <= region[regionCounter]-1)
150             {
151                 pose reachedPoint;
152                 reachedPoint.x = status->point.x;
153                 reachedPoint.y = status->point.y;
154                 newPose = getComponent(furCoordCD,
155                 getIndex(furCoordCD,reachedPoint)-1);
156
157                 reach.point.x = newPose.x;
158                 reach.point.y = newPose.y;
159                 reach.hint = "cd_stage1";
160                 reach.status = false;
161                 reach_point_pub.publish(reach);
162             }
163             //If number of processed tracks is bigger than
164             //available tracks in that region then go home
165             else if (numOfProcessedTracks > region[regionCounter]-1)
166             {
```

```
167                 if (rank == 1)
168                 {
169                     pose newPose;
170                     newPose.x = hX;
171                     newPose.y = hY;
172                     reach.point.x = newPose.x;
173                     reach.point.y = newPose.y;
174                     reach.hint = "home";
175                     reach.status = false;
176                     reach_point_pub.publish(reach);
177                 }
178                 else if (rank != 1)
179                 {
180                     pose reachedPoint;
181                     reachedPoint.x = status->point.x;
182                     reachedPoint.y = status->point.y;
183                     newPose = getComponent(homeCD,
184                     getIndex(homeCD,reachedPoint)-1);
185 
186                     reach.point.x = newPose.x;
187                     reach.point.y = newPose.y;
188                     reach.hint = "home_cd";
189                     reach.status = false;
190                     reach_point_pub.publish(reach);
191                 }
192             }
193         }
194 
195         if (status->hint.compare("cd_stage1") == 0)
196         {
197             //No checking is required as you are sure
198             //that next round is still available
199             pose reachedPoint;
200             reachedPoint.x = status->point.x;
201             reachedPoint.y = status->point.y;
202             newPose = getComponent(furCoordAB,
203             getIndex(furCoordCD,reachedPoint));
204 
205             reach.point.x = newPose.x;
206             reach.point.y = newPose.y;
207             reach.hint = "ab";
208             reach.status = false;
209             reach_point_pub.publish(reach);
210             numOfProcessedTracks++;
211         }
212         if (status->hint.compare("ab") == 0)
213         {
214             //If number of processed tracks is smaller than number
215             //of tracks in your region then transit your track
216             if (numOfProcessedTracks <= region[regionCounter]-1)
217             {
218                 pose reachedPoint;
219                 reachedPoint.x = status->point.x;
220                 reachedPoint.y = status->point.y;
221                 newPose = getComponent(furCoordAB,
222                 getIndex(furCoordAB,reachedPoint)-1);
223 
224                 reach.point.x = newPose.x;
225                 reach.point.y = newPose.y;
226                 reach.hint = "ab_stage1";
```

```
227                 reach.status = false;
228                 reach_point_pub.publish(reach);
229             }
230             //If number of processed tracks is bigger than
231             //available tracks in that region then go home
232             else if (numOfProcessedTracks > region[regionCounter]-1)
233             {
234                 if (rank == 1)
235                 {
236                     pose newPose;
237                     newPose.x = hX;
238                     newPose.y = hY;
239                     reach.point.x = newPose.x;
240                     reach.point.y = newPose.y;
241                     reach.hint = "home";
242                     reach.status = false;
243                     reach_point_pub.publish(reach);
244                 }
245                 else if (rank != 1)
246                 {
247                     pose reachedPoint;
248                     reachedPoint.x = status->point.x;
249                     reachedPoint.y = status->point.y;
250                     newPose = getComponent(homeAB,
251                     getIndex(homeAB,reachedPoint)-1);
252
253                     reach.point.x = newPose.x;
254                     reach.point.y = newPose.y;
255                     reach.hint = "home_ab";
256                     reach.status = false;
257                     reach_point_pub.publish(reach);
258                 }
259             }
260         }
261
262         if (status->hint.compare("ab_stage1") == 0)
263         {
264             //No checking is required as you are
265             //sure that next round is still available
266             pose reachedPoint;
267             reachedPoint.x = status->point.x;
268             reachedPoint.y = status->point.y;
269             newPose = getComponent(furCoordCD,
270             getIndex(furCoordAB,reachedPoint));
271
272             reach.point.x = newPose.x;
273             reach.point.y = newPose.y;
274             reach.hint = "cd";
275             reach.status = false;
276             reach_point_pub.publish(reach);
277             numOfProcessedTracks++;
278         }
279         //Going back to home position through cd sideline
280         if (status->hint.compare("home_cd") == 0)
281         {
282             pose reachedPoint;
283             reachedPoint.x = status->point.x;
284             reachedPoint.y = status->point.y;
285             if (getIndex(homeCD,reachedPoint)-1 > -1)
286             {
```

```
287                 newPose = getComponent(homeCD,
288                 getIndex(homeCD,reachedPoint)-1);
289
290                 reach.point.x = newPose.x;
291                 reach.point.y = newPose.y;
292                 reach.hint = "home_cd";
293                 reach.status = false;
294                 reach_point_pub.publish(reach);
295             }
296             if (getIndex(homeCD,reachedPoint)-1 == -1)
297             {
298                 reach.point.x = hX;
299                 reach.point.y = hY;
300                 reach.hint = "home";
301                 reach.status = false;
302                 reach_point_pub.publish(reach);
303             }
304         }
305         //Going back to home position through ab sideline
306         if (status->hint.compare("home_ab") == 0)
307         {
308             pose reachedPoint;
309             reachedPoint.x = status->point.x;
310             reachedPoint.y = status->point.y;
311             if (getIndex(homeAB,reachedPoint)-1 > -1)
312             {
313                 newPose = getComponent(homeAB,
314                 getIndex(homeAB,reachedPoint)-1);
315
316                 reach.point.x = newPose.x;
317                 reach.point.y = newPose.y;
318                 reach.hint = "home_ab";
319                 reach.status = false;
320                 reach_point_pub.publish(reach);
321             }
322             if (getIndex(homeAB,reachedPoint)-1 == -1)
323             {
324                 reach.point.x = hX;
325                 reach.point.y = hY;
326                 reach.hint = "home";
327                 reach.status = false;
328                 reach_point_pub.publish(reach);
329             }
330         }
331     }
332     //robot will be confident whenever it finds an unploughed furrow
333     else if (!iAmConfident)
334     {
335         //Step 1: Find your location respect to alpha
336           float regAngle = atan2(currentPose.y-alpha.y,
337           currentPose.x-alpha.x)*180/PI;
338
339           if(pointName.compare("beta") == 0 ||
340           pointName.compare("psi") == 0)
341
342           {
343             reach.point.x = alpha.x;
344             reach.point.y = alpha.y;
345             reach.hint = "alpha";
346             reach.status = false;
```

```
347             reach_point_pub.publish(reach);
348           }
349           else if (pointName.compare("rho") == 0)
350           {
351               reach.point.x = gama.x;
352               reach.point.y = gama.y;
353               reach.hint = "gama";
354               reach.status = false;
355               reach_point_pub.publish(reach);
356           }
357           else if(pointName.compare("gama") == 0)
358           {
359             reach.point.x = beta.x;
360             reach.point.y = beta.y;
361             reach.hint = "beta";
362             reach.status = false;
363             reach_point_pub.publish(reach);
364           }
365
366
367         if (pointName.compare("ab_reg") == 0)
368         {
369             //if the requested point is occupied
370             if (targetIsOccupied)
371             {
372                 //update your rank
373                 rank++;
374                 regionCounter++;
375                 //go to the next ploughing target
376                 newPose = regPoint[regionCounter];
377                 reach.point.x = newPose.x;
378                 reach.point.y = newPose.y-tranY;
379                 reach.hint = "ab_reg";
380                 reach.status = true;
381                 reach_point_pub.publish(reach);
382             }
383             //if the requested point is not occupied
384             else if (!targetIsOccupied)
385             {
386                 //start ploughing
387                 pose newPose = getComponent(furCoordAB,getIndex
388                 (furCoordAB,regPoint[regionCounter]));
389                 reach.point.x = newPose.x;
390                 reach.point.y = newPose.y;
391                 if (rank == teamSize)
392                 {
393                     reach.hint = "ab_start";
394                 }
395                 else if (rank != teamSize)
396                 {
397                     reach.hint = "ab_conf";
398                 }
399                 reach.status = false;
400                 reach_point_pub.publish(reach);
401                 iAmConfident = true;
402             }
403         }
404     }
405 }
406
```

```
407 void TaskHandler::imgCB(const std_msgs::Bool::ConstPtr &imgStatus)
408 {
409     targetIsOccupied = imgStatus->data;
410 }
411 //a method to receive the task command
412 void TaskHandler::taskCmdCallback
413 (const ploughing_reversible::task::ConstPtr &msg)
414 {
415     //loading data to appropriate variables
416     taskCmd = msg->task;
417     alpha.x = msg->alpha.x;
418     alpha.y = msg->alpha.y;
419     beta.x = msg->beta.x;
420     beta.y = msg->beta.y;
421     gama.x = msg->gamma.x;
422     gama.y = msg->gamma.y;
423     psi.x = msg->psi.x;
424     psi.y = msg->psi.y;
425     rho.x = msg->rho.x;
426     rho.y = msg->rho.y;
427     A.x = msg->a.x;
428     A.y = msg->a.y;
429     B.x = msg->b.x;
430     B.y = msg->b.y;
431     C.x = msg->c.x;
432     C.y = msg->c.y;
433     D.x = msg->d.x;
434     D.y = msg->d.y;
435     furDist = msg->furrow_distance;
436     teamSize = msg->team_size;
437     // calculate furrow coordinates
438     furCoordAB = furrowLocalizer(A,B,furDist);
439     furCoordCD = furrowLocalizer(C,D,furDist);
440     furTotal = furCoordCD.size();
441     //calculating homing coordinates
442     C.y = C.y + tranY;
443     D.y = D.y + tranY;
444     A.y = A.y - tranY;
445     B.y = B.y - tranY;
446     homeCD = furrowLocalizer(C,D,furDist);
447     homeAB = furrowLocalizer(A,B,furDist);
448     //Region Division
449     //First resize "region" and "regPoint" and
450     //"furNum" vector to suit the team size.
451     region.resize(teamSize);
452     regPoint.resize(teamSize);
453     furNum.resize(teamSize);
454     //Then allocate number of tracks to each region.
455     //Vectors count from 0 therefore teamsize-1.
456     for (int i=0;i<=teamSize-1;i++)
457     {
458         region[i] = floor(furTotal/teamSize);
459         if (furTotal%teamSize > i)
460         {
461             region[i]++;
462         }
463         furNum[i] = furNum[i-1]+region[i];
464         regPoint[i] = getComponent(furCoordAB,furNum[i]-1);
465     }
466     //Step 1: Find your location respect to alpha
```

```
467
468     float regAngle = atan2(currentPose.y-alpha.y,
469     currentPose.x-alpha.x)*180/PI;
470
471     if(regAngle>= -1 && regAngle <= 90.0)
472     {
473       reach.point.x = alpha.x;
474       reach.point.y = alpha.y;
475       reach.hint = "alpha";
476       reach.status = false;
477       reach_point_pub.publish(reach);
478     }
479     else if(regAngle> 90.0 && regAngle <= 179.0)
480     {
481       //Check if you are above D.y or below D.y
482       if (currentPose.y >= D.y)
483       {
484         reach.point.x = psi.x;
485         reach.point.y = psi.y;
486         reach.hint = "psi";
487         reach.status = false;
488         reach_point_pub.publish(reach);
489       }
490       if (currentPose.y < D.y)
491       {
492         reach.point.x = rho.x;
493         reach.point.y = rho.y;
494         reach.hint = "rho";
495         reach.status = false;
496         reach_point_pub.publish(reach);
497       }
498
499     }
500     else if(regAngle < -1 && regAngle >= -90.0)
501     {
502       reach.point.x = beta.x;
503       reach.point.y = beta.y;
504       reach.hint = "beta";
505       reach.status = false;
506       reach_point_pub.publish(reach);
507     }
508     else if(regAngle < -90.0 && regAngle >= -179.0)
509     {
510       reach.point.x = gama.x;
511       reach.point.y = gama.y;
512       reach.hint = "gama";
513       reach.status = false;
514       reach_point_pub.publish(reach);
515     }
516     //Step 2: approach the selected location
517
518 }
519 void TaskHandler::poseCB(const nav_msgs::Odometry::ConstPtr &pose)
520 {
521   currentPose.x = pose->pose.pose.position.x;
522   currentPose.y = pose->pose.pose.position.y;
523 }
524 //main method
525 int main(int argc, char** argv)
526 {
```

```
527     ros::init(argc,argv,"s1_th");
528     TaskHandler th;
529     ros::spin();
530     return 0;
531 }
532 //a method to map the furrow coordinates.
533 //it collects all calculated coordinates in a list.
534 list<pose> furrowLocalizer(pose start, pose end, float dist)
535 {
536     list<pose> createdList;
537     pose currentPose = start;
538     while(currentPose.x >= end.x)
539     {
540         /*
541             push_back -> add element at the end, last element.
542             push_front -> add element to the front, element 0.
543         */
544         createdList.push_back(currentPose);
545         currentPose.x -= dist;
546     }
547     return createdList;
548 }
549 //a method to get particular component in a given list.
550 pose getComponent(list<pose> _list, int _i){
551     list<pose>::iterator it = _list.begin();
552     for(int i=0; i<_i; i++){
553         ++it;
554     }
555     return *it;
556 }
557 int getIndex(list<pose> _list, pose _pose)
558 {
559     int index = -1;
560     pose temp;
561     for(int i=0; i<_list.size(); i++)
562     {
563         temp = getComponent(_list,i);
564         if (temp.x == _pose.x && temp.y == _pose.y)
565         {
566             index = i;
567         }
568     }
569     return index;
570 }
```

Appendix K

C++ Code for Reach Point Using ROS

```
1  #include <ros/ros.h>
2  #include <tf/tf.h>
3  #include <sensor_msgs/LaserScan.h>
4  #include <ploughing_reversible/ReachPoint.h>
5  #include <nav_msgs/Odometry.h>
6  #include <geometry_msgs/Twist.h>
7  #include <geometry_msgs/Quaternion.h>
8  #include <string>
9  #include <pthread.h>
10 #include <math.h>
11 #include <sstream>
12 #include <stdlib.h>     /* srand, rand */
13 #include <time.h>
14
15
16 #define PI 3.14159265359
17 #define Rad2Deg 57.2957795
18 #define Deg2Rad 0.0174532925
19 #define PITCH 0.3522504892367
20 #define DistanceTh 0.2
21 #define DistanceCrt 1.5
22 #define DistanceCrtTwo 0.5
23 #define MaxVel 0.5
24 #define AlgorithmEffectDistance 5.5
25
26 using namespace std;
27
28 struct force
29 {
30   float mag;
31   float theta;
32   float x;
33   float y;
34 };
35
36 struct pose
37 {
38   float x;
39   float y;
40   double theta;
41 };
42 struct mapMem
43 {
44   float Rx;
45   float Ry;
46   float Ox;
```

```
47    float Oy;
48    int time;
49    float lr;
50    float lr_ang;
51    float lrx;
52    float lry;
53  };
54  mapMem fmm_old;
55  int fjcount = 0;
56  float fjmean= 0.0;
57  pose current, ab[10];
58
59  void stop(void);
60  void *move(void* threadID);
61  geometry_msgs::Twist vel;
62  ploughing_reversible::ReachPoint target,reqPoint, prevPoint;
63  pose currentGlobal, requestedTarget;
64  bool targetIsReached = false, headingIsCorrected = false;
65  pthread_t moveThHandler_;
66  string procPointHint="";
67  int robot_id = 0;
68  float dl = 0.5, df = 3.0, dr = 1.5;
69  float reversCoef = 1.0;
70  ros::Publisher vel_pub;
71  ros::Publisher target_status_pub;
72
73
74  class ReachPoint
75  {
76    public:
77      ReachPoint();
78    private:
79      ros::NodeHandle nh;
80      ros::Subscriber target_sub;
81      ros::Subscriber laser_sub;
82      ros::Subscriber world_pose_sub;
83
84      void laserCallBack(const sensor_msgs::
85      LaserScan::ConstPtr &scan);
86
87      void worldPoseCallback(const nav_msgs::
88      Odometry::ConstPtr &world_pose);
89
90      void targetCallback(const ploughing_reversible::
91      ReachPoint::ConstPtr &target);
92  };
93
94  ReachPoint::ReachPoint()
95  {
96    ros::NodeHandle n("~");
97    n.getParam("robot_id",robot_id);
98    ostringstream ss;
99    ss << "robot_";
100   ss << robot_id;
101
102   string robot_name = ss.str();
103   string laser_pipename = robot_name + "/base_scan";
104   string pose_pipename = robot_name + "/base_pose_ground_truth";
105   string twist_pipename = robot_name + "/cmd_vel";
106   string pointreq_pipename = robot_name + "/requested_point";
```

```
107     string pointres_pointname = robot_name + "/reach_point_status";
108
109     target_sub = nh.subscribe<ploughing_reversible::ReachPoint>
110     (pointreq_pipename,5,&ReachPoint::targetCallback,this);
111
112     laser_sub = nh.subscribe<sensor_msgs::LaserScan>
113     (laser_pipename,5,&ReachPoint::laserCallBack,this);
114
115     world_pose_sub = nh.subscribe<nav_msgs::Odometry>
116     (pose_pipename,5,&ReachPoint::worldPoseCallback,this);
117
118     vel_pub = nh.advertise<geometry_msgs::Twist>
119     (twist_pipename,1);
120
121     target_status_pub = nh.advertise<ploughing_reversible::ReachPoint>
122     (pointres_pointname,5);
123
124     for (int i=0; i<=9; i++)
125     {
126       ab[i].x = 10 - AlgorithmEffectDistance - i*1;
127       ab[i].y = -13;
128     }
129
130
131   }
132   //a method to receive laser range finder readings.
133   void ReachPoint::laserCallBack
134   (const sensor_msgs::LaserScan::ConstPtr &scan)
135   {
136     bool roadIsBlocked = false,
137          rightIsBlocked = false,
138          leftIsBlocked = false;
139     float frontObstacleMax = 4.5,
140           FOBMaxAngle = 0.0,
141           leftObstacleMax = 4.5,
142           LOBMaxAngle = 0.0,
143           rightObstacleMax = 4.5,
144           ROBMaxAngle = 0.0;
145
146     pose dir;
147
148     if (reqPoint.hint.compare("done") == 0)
149     {
150       //For APF
151       float beams[512];
152       pose obj_new[512], obj[512];
153       force attraction, repulsion[512], sum;
154       int RotCoef = 1;
155       sum.x = 0.0;
156       sum.y = 0.0;
157       int beamCounter = 1;
158       dir.x = requestedTarget.x - currentGlobal.x;
159       dir.y = requestedTarget.y - currentGlobal.y;
160       dir.theta = atan2(dir.y,dir.x)*Rad2Deg;
161       float distanceToTarget = sqrt(dir.x*dir.x + dir.y*dir.y);
162
163       float R[2][2] = {{cos((90-currentGlobal.theta)*Deg2Rad),
164                         -1*sin((90-currentGlobal.theta)*Deg2Rad)},
165                        {sin((90-currentGlobal.theta)*Deg2Rad),
166                         cos((90-currentGlobal.theta)*Deg2Rad)}};
```

```
167
168       if (distanceToTarget >= 3.0)
169       {
170         for (int i=0; i<=511; i++)
171         {
172           if (scan->ranges[i] < 4.5 )
173           {
174             beams[i] = scan->ranges[i];
175
176             obj[i].x = beams[i]*cos((i*PITCH)*Deg2Rad);
177             obj[i].y = beams[i]*sin((i*PITCH)*Deg2Rad);
178             beamCounter++;
179           }
180           else if (scan->ranges[i] == 4.5)
181           {
182             beams[i] = 0.0;
183             obj[i].x = 0.0;
184             obj[i].y = 0.0;
185           }
186         }
187
188         //form the orientation matrix
189         for (int i=0; i<= 511; i++)
190         {
191           if (beams[i] > 0.0 && beams[i] < 4.5)
192           {
193
194             repulsion[i].x = obj[i].x;
195             repulsion[i].y = obj[i].y;
196
197             //remember: repulsion force is 180 degrees mirrored
198             //of the vector describing the object and the robot
199             if (beams[i] <= DistanceCrt && beams[i] > DistanceCrtTwo)
200             {
201               repulsion[i].x = 20*obj[i].x;
202               repulsion[i].y = 20*obj[i].y;
203               RotCoef = 10;
204             }
205             if (beams[i] <= DistanceCrtTwo)
206             {
207               repulsion[i].x = 100*obj[i].x;
208               repulsion[i].y = 100*obj[i].y;
209               RotCoef = 100;
210             }
211             if (beams[i] > DistanceCrt)
212             {
213               repulsion[i].x = obj[i].x;
214               repulsion[i].y = obj[i].y;
215               RotCoef = 1;
216             }
217             repulsion[i].mag = 1/sqrt(repulsion[i].x*repulsion[i].x
218             +repulsion[i].y*repulsion[i].y);
219
220             repulsion[i].theta = atan2(repulsion[i].y,repulsion[i].x);
221           }
222           else if (beams[i] == 0.0)
223           {
224             repulsion[i].x = 0.0;
225             repulsion[i].y = 0.0;
226             repulsion[i].mag = 0.0;
```

```
227             repulsion[i].theta = 0.0;
228           }
229
230           sum.x = sum.x + repulsion[i].x;
231           sum.y = sum.y + repulsion[i].y;
232         }
233
234
235         float rx = sum.x*cos(180*Deg2Rad) - sum.y*sin(180*Deg2Rad);
236         float ry = sum.x*sin(180*Deg2Rad) + sum.y*cos(180*Deg2Rad);
237         sum.x = rx*R[0][0] - ry*R[0][1];
238         sum.y = -rx*R[1][0] + ry*R[1][1];
239         sum.mag = sqrt(sum.x*sum.x+sum.y*sum.y);
240         sum.theta = Rad2Deg*atan2(sum.y,sum.x);
241
242         //attraction force
243         attraction.mag = sqrt(dir.x*dir.x+dir.y*dir.y);
244         attraction.theta = Rad2Deg*atan2(dir.y,dir.x);
245         attraction.x = dir.x;
246         attraction.y = dir.y;
247
248         //sum force
249         int Krep = 7;
250         int Katt = 20;
251         sum.x = sum.x + Katt*attraction.x;
252         sum.y = sum.y + Katt*attraction.y;
253         sum.theta = Rad2Deg*atan2(sum.y,sum.x);
254         sum.mag = sqrt(sum.x*sum.x+sum.y*sum.y);
255
256         /*Here is where you assign speed to the motors*/
257         if (abs(sum.theta-currentGlobal.theta) < 180)
258         {
259           vel.angular.z = RotCoef*0.02*(sum.theta-currentGlobal.theta);
260         }
261
262         else if ((sum.theta-currentGlobal.theta > 180) ||
263         (sum.theta-currentGlobal.theta < -180))
264         {
265           vel.angular.z = RotCoef*-0.02*(sum.theta-currentGlobal.theta);
266         }
267
268         vel.linear.x = sum.mag;
269         //limiting the linear speed to 0.75 m/s
270         if (vel.linear.x >= MaxVel)
271         {
272           vel.linear.x = MaxVel;
273         }
274         if (vel.linear.x <= -1*MaxVel)
275         {
276           vel.linear.x = -1*MaxVel;
277         }
278         vel_pub.publish(vel);
279
280       }
281       else if (distanceToTarget < 3.0)
282       {
283         stop();
284         reqPoint.status = true;
285         prevPoint.hint = reqPoint.hint;
286         target_status_pub.publish(reqPoint);
```

```
287         procPointHint = reqPoint.hint;
288       }
289     }
290     /*check the "reqPoint.hint" to assign limitation on field of view*/
291     else if (reqPoint.hint.compare("done") != 0)
292     {
293         dir.x = requestedTarget.x - currentGlobal.x;
294         dir.y = requestedTarget.y - currentGlobal.y;
295 
296         if ((reqPoint.hint.compare("ab1") == 0 &&
297            sqrt(dir.x*dir.x + dir.y*dir.y) >= AlgorithmEffectDistance) ||
298             (reqPoint.hint.compare("ab1-left") == 0 &&
299             sqrt(dir.x*dir.x + dir.y*dir.y) >= AlgorithmEffectDistance - 1.5) ||
300             (reqPoint.hint.compare("ab1-adjust") == 0 &&
301             sqrt(dir.x*dir.x + dir.y*dir.y) >= AlgorithmEffectDistance - 1.5) ||
302             (reqPoint.hint.compare("ab-resume") == 0 &&
303             sqrt(dir.x*dir.x + dir.y*dir.y) >= AlgorithmEffectDistance - 1.0))
304         {
305           dir.x = requestedTarget.x - currentGlobal.x;
306           dir.y = requestedTarget.y - currentGlobal.y;
307           dir.theta = atan2(dir.y,dir.x)*Rad2Deg;
308           float distanceToTarget = sqrt(dir.x*dir.x + dir.y*dir.y);
309           //For APF
310           float beams[512];
311           pose dir, obj_new[512], obj[512];
312           force attraction, repulsion[512], sum;
313           int RotCoef = 1;
314           sum.x = 0.0;
315           sum.y = 0.0;
316           int beamCounter = 1;
317           dir.x = requestedTarget.x - currentGlobal.x;
318           dir.y = requestedTarget.y - currentGlobal.y;
319           dir.theta = atan2(dir.y,dir.x)*Rad2Deg;
320           distanceToTarget = sqrt(dir.x*dir.x + dir.y*dir.y);
321           float R[2][2] = {{cos((90-currentGlobal.theta)*Deg2Rad),
322                             -1*sin((90-currentGlobal.theta)*Deg2Rad)},
323                             {sin((90-currentGlobal.theta)*Deg2Rad),
324                             cos((90-currentGlobal.theta)*Deg2Rad)}};
325           if (distanceToTarget >= DistanceTh)
326           {
327             for (int i=0; i<=511; i++)
328             {
329               if (scan->ranges[i] < 4.5 )
330               {
331                 beams[i] = scan->ranges[i];
332 
333                 obj[i].x = beams[i]*cos((i*PITCH)*Deg2Rad);
334                 obj[i].y = beams[i]*sin((i*PITCH)*Deg2Rad);
335                 beamCounter++;
336               }
337               else if (scan->ranges[i] == 4.5)
338               {
339                 beams[i] = 0.0;
340                 obj[i].x = 0.0;
341                 obj[i].y = 0.0;
342               }
343             }
344 
345             //form the orientation matrix
346             for (int i=0; i<= 511; i++)
```

```
347             {
348               if (beams[i] > 0.0 && beams[i] < 4.5)
349               {
350
351                 repulsion[i].x = obj[i].x;
352                 repulsion[i].y = obj[i].y;
353
354                 //remember: repulsion force is 180 degrees
355                 //mirrored of the vector describing the object and the robot
356                 if (beams[i] <= DistanceCrt && beams[i] > DistanceCrtTwo)
357                 {
358                   repulsion[i].x = 20*obj[i].x;
359                   repulsion[i].y = 20*obj[i].y;
360                   RotCoef = 10;
361                 }
362                 if (beams[i] <= DistanceCrtTwo)
363                 {
364                   repulsion[i].x = 100*obj[i].x;
365                   repulsion[i].y = 100*obj[i].y;
366                   RotCoef = 100;
367                 }
368                 if (beams[i] > DistanceCrt)
369                 {
370                   repulsion[i].x = obj[i].x;
371                   repulsion[i].y = obj[i].y;
372                   RotCoef = 1;
373                 }
374                 repulsion[i].mag = 1/sqrt(repulsion[i].x*repulsion[i].x
375                 +repulsion[i].y*repulsion[i].y);
376
377                 repulsion[i].theta = atan2(repulsion[i].y,repulsion[i].x);
378               }
379               else if (beams[i] == 0.0)
380               {
381                 repulsion[i].x = 0.0;
382                 repulsion[i].y = 0.0;
383                 repulsion[i].mag = 0.0;
384                 repulsion[i].theta = 0.0;
385               }
386
387               sum.x = sum.x + repulsion[i].x;
388               sum.y = sum.y + repulsion[i].y;
389
390             }
391             float rx = sum.x*cos(180*Deg2Rad) - sum.y*sin(180*Deg2Rad);
392             float ry = sum.x*sin(180*Deg2Rad) + sum.y*cos(180*Deg2Rad);
393             sum.x = rx*R[0][0] - ry*R[0][1];
394             sum.y = -rx*R[1][0] + ry*R[1][1];
395
396
397             //Applying Repulsion Force From the Field
398             if (currentGlobal.y < -15.0 && (-15.0 - currentGlobal.y <= 2.0)
399             && currentGlobal.x <= 11.0
400             && currentGlobal.x >= -11.0)
401             {
402               sum.x += 10000.00;
403               sum.y += -10000.00;
404             }
405             else if (currentGlobal.y < 13.0 && currentGlobal.y > -13
406             && currentGlobal.x >= 12.0
```

```
407             && (currentGlobal.x - 12 <= 0.75))
408             {
409               sum.x += 10000.00;
410               sum.y += 10000.00;
411             }
412             else if (currentGlobal.y > 12.0
413             && (currentGlobal.y - 13.0 <= 2.0)
414             && currentGlobal.x <= 11.0 && currentGlobal.x > 0.0)
415             {
416               sum.x += -10000.0;
417               sum.y += 10000.00;
418             }
419             else if(currentGlobal.y > 12.0
420             && (currentGlobal.y - 13 <= 2.0)
421             && currentGlobal.x <= 0.0
422             && currentGlobal.x >= -11.0)
423             {
424               sum.x += 10000.00;
425               sum.y += 10000.00;
426             }
427             else if (currentGlobal.y < 13.0
428             && currentGlobal.y > -13.0
429             && currentGlobal.x <= -12.0
430             && ( -12 - currentGlobal.x <= 0.75))
431             {
432               sum.x += -10000.00;
433               sum.y += 10000.00;
434             }
435
436             sum.mag = sqrt(sum.x*sum.x+sum.y*sum.y);
437             sum.theta = Rad2Deg*atan2(sum.y,sum.x);
438             //attraction force
439             attraction.mag = sqrt(dir.x*dir.x+dir.y*dir.y);
440             attraction.theta = Rad2Deg*atan2(dir.y,dir.x);
441             attraction.x = dir.x;
442             attraction.y = dir.y;
443             //sum force
444             int Krep = 7;
445             int Katt = 20;
446             sum.x = sum.x + Katt*attraction.x;
447             sum.y = sum.y + Katt*attraction.y;
448             sum.theta = Rad2Deg*atan2(sum.y,sum.x);
449             sum.mag = sqrt(sum.x*sum.x+sum.y*sum.y);
450             /*Here is where you assign speed to the motors*/
451             if (abs(sum.theta-currentGlobal.theta) < 180)
452             {
453               vel.angular.z = RotCoef*0.02*(sum.theta-currentGlobal.theta);
454             }
455             else if ((sum.theta-currentGlobal.theta > 180)
456             || (sum.theta-currentGlobal.theta < -180))
457             {
458               vel.angular.z = RotCoef*-0.02*(sum.theta-currentGlobal.theta);
459             }
460             vel.linear.x = sum.mag;
461             //limiting the linear speed to 0.75 m/s
462             if (vel.linear.x >= MaxVel)
463             {
464               vel.linear.x = MaxVel;
465             }
466             if (vel.linear.x <= -1*MaxVel)
```

```
467             {
468               vel.linear.x = -1*MaxVel;
469             }
470             vel_pub.publish(vel);
471
472           }
473           else if (distanceToTarget <DistanceTh)
474           {
475             stop();
476             reqPoint.status = true;
477             prevPoint.hint = reqPoint.hint;
478             target_status_pub.publish(reqPoint);
479             procPointHint = reqPoint.hint;
480           }
481         }
482         else
483         {
484             for(int i=191; i<=319; i++)
485             {
486               if(scan->ranges[i] <= df)
487               {
488                 roadIsBlocked = true;
489                 if (scan->ranges[i] < frontObstacleMax)
490                 {
491                   frontObstacleMax = scan->ranges[i];
492                   FOBMaxAngle = i*PITCH*PI/180;
493                 }
494               }
495             }
496             for(int j=0; j<=127; j++)
497             {
498               if(scan->ranges[j] <= dr)
499               {
500                 rightIsBlocked = true;
501                 if (scan->ranges[j] < rightObstacleMax)
502                 {
503                   rightObstacleMax = scan->ranges[j];
504                   ROBMaxAngle = j*PITCH*PI/180;
505                 }
506               }
507             }
508             for(int k=385; k<=512; k++)
509             {
510               if(scan->ranges[k] <= dl)
511               {
512                 leftIsBlocked = true;
513                 if (scan->ranges[k] < leftObstacleMax)
514                 {
515                   leftObstacleMax = scan->ranges[k];
516                   LOBMaxAngle = k*PITCH*PI/180;
517                 }
518               }
519             }
520           //}
521           if (reqPoint.hint.compare("ab1") != 0
522               && reqPoint.hint.compare("ab1-adjust") != 0
523               && reqPoint.hint.compare("ab1-left") != 0
524               && reqPoint.hint.compare("ab-resume") != 0)
525           {
526             rightIsBlocked = false;
```

```
527             leftIsBlocked = false;
528           }
529
530         /*this is a section which guides the robot,
531           it has been put in here because this function
532           is being updated 5 times a second.
533           Remeber because is being called very often,
534           we are not allowed to put any while loop in here, if we
535           do that, laser range finder readings will be missed.
536         */
537           pose dir, prevdir;
538           dir.x = requestedTarget.x - currentGlobal.x;
539           dir.y = requestedTarget.y - currentGlobal.y;
540           dir.theta = atan2(dir.y,dir.x)*Rad2Deg;
541           prevdir = dir;
542
543           if(abs(dir.theta-currentGlobal.theta) >= 10.0
544           && (reqPoint.hint.compare("cd") == 0
545           || reqPoint.hint.compare("ab") == 0)
546           && prevPoint.hint.compare(reqPoint.hint.c_str()) != 0)
547           {
548             //heading correction
549             if (abs(dir.theta-currentGlobal.theta) < 180)
550             {
551               vel.angular.z = 0.04*(dir.theta-currentGlobal.theta);
552             }
553             else if ((dir.theta-currentGlobal.theta > 180)
554             || (dir.theta-currentGlobal.theta < -180))
555             {
556               vel.angular.z = -0.04*(dir.theta-currentGlobal.theta);
557             }
558             vel.linear.x = 0.0;
559             vel_pub.publish(vel);
560             headingIsCorrected = false;
561           }
562           else if (reqPoint.hint.compare("cd") != 0 ||
563           reqPoint.hint.compare("ab") != 0)
564           {
565             headingIsCorrected = true;
566           }
567             if(!targetIsReached && headingIsCorrected)
568             {
569               //Congestion avoidance
570               if ((roadIsBlocked || rightIsBlocked) &&
571               sqrt(dir.x*dir.x + dir.y*dir.y) <=
572               frontObstacleMax*sin(FOBMaxAngle)/2)
573               {
574                 roadIsBlocked = false;
575                 rightIsBlocked = false;
576               }
577               //if lasers didn't detect anything in the field,
578               //approach the target.
579               if(!roadIsBlocked && !rightIsBlocked
580               && !leftIsBlocked)
581               {
582                 //update the position of the robot respect to target
583                 dir.x = requestedTarget.x - currentGlobal.x;
584                 dir.y = requestedTarget.y - currentGlobal.y;
585                 dir.theta = atan2(dir.y,dir.x)*Rad2Deg;
586                 //simulation cannot reach 180 or -180 degrees.
```

```
587
588             if (abs(dir.theta-currentGlobal.theta) < 180)
589             {
590               vel.angular.z = 0.02*(dir.theta-currentGlobal.theta);
591             }
592             else if ((dir.theta-currentGlobal.theta > 180)
593             || (dir.theta-currentGlobal.theta < -180))
594             {
595               vel.angular.z = -0.02*(dir.theta-currentGlobal.theta);
596             }
597             vel.linear.x = sqrt(dir.x*dir.x + dir.y*dir.y);
598             //limiting the linear speed to 0.75 m/s
599             if (vel.linear.x >= 0.75)
600             {
601               vel.linear.x = 0.75;
602             }
603             if (vel.linear.x <= -0.75)
604             {
605               vel.linear.x = -0.75;
606             }
607             vel_pub.publish(vel);
608             prevdir = dir;
609           }
610           //if any obstacle is detected within the
611           //field of view of the robot
612         else if(roadIsBlocked || rightIsBlocked || leftIsBlocked)
613         {
614             if (reqPoint.hint.compare("ab1") == 0
615             || reqPoint.hint.compare("ab1-adjust") == 0
616             || reqPoint.hint.compare("ab1-left") == 0
617             || reqPoint.hint.compare("ab-resume") == 0)
618             {
619                   reversCoef = rand() % 20 + 1;
620                   vel.linear.x = -1*reversCoef*0.02;
621                   vel.angular.z = 0.0;
622                   vel_pub.publish(vel);
623                   prevdir = dir;
624             }
625             else if (reqPoint.hint.compare("cd") == 0
626             || reqPoint.hint.compare("ab") == 0)
627             {
628               vel.linear.x = 0.0;
629               vel.angular.z = 0.0;
630               vel_pub.publish(vel);
631             }
632           }
633         }
634
635       //if we have reached in distance of 5 cm or less of the target,
636       //target is assumed to be reached.
637       if (sqrt(dir.x*dir.x + dir.y*dir.y) <= 0.20)
638       {
639           targetIsReached = true;
640           stop();
641           reqPoint.status = true;
642           prevPoint.hint = reqPoint.hint;
643           target_status_pub.publish(reqPoint);
644           procPointHint = reqPoint.hint;
645       }
646     }
```

```
647       }
648   }
649
650   //a method to receive global position of the robot.
651   void ReachPoint::worldPoseCallback
652   (const nav_msgs::Odometry::ConstPtr &world_pose)
653   {
654     currentGlobal.x = world_pose->pose.pose.position.x;
655     currentGlobal.y = world_pose->pose.pose.position.y;
656
657     currentGlobal.theta = Rad2Deg *
658     tf::getYaw(world_pose->pose.pose.orientation);
659   }
660   //a method to receive target coordinates.
661   void ReachPoint::targetCallback
662   (const ploughing_reversible::ReachPoint::ConstPtr &target)
663   {
664     //keeping record of the requested target
665     reqPoint.point.x = target->point.x;
666     reqPoint.point.y = target->point.y;
667     reqPoint.hint = target->hint;
668     reqPoint.status = target->status;
669     //loading the target
670     requestedTarget.x = target->point.x;
671     requestedTarget.y = target->point.y;
672
673     if(reqPoint.hint.compare("ab1") == 0)
674     {
675       df = 1.5;
676       dl = 0.5;
677       dr = 1.5;
678     }
679     else if(reqPoint.hint.compare("ab1-adjust") == 0)
680     {
681       df = 3.0;
682       dl = 1.0;
683       dr = 3.0;
684     }
685     else if(reqPoint.hint.compare("ab") == 0)
686     {
687       df = 0.5;
688       dl = 0.0;
689       dr = 0.0;
690     }
691     else if(reqPoint.hint.compare("cd") == 0 )
692     {
693       df = 1.0;
694       dl = 0.0;
695       dr = 0.0;
696     }
697     else if(reqPoint.hint.compare("ab1-left") == 0
698    || reqPoint.hint.compare("done") == 0)
699     {
700       df = 1.0;
701       dl = 0.5;
702       dr = 0.75;
703     }
704     targetIsReached = false;
705     headingIsCorrected = false;
706   }
```

```
707
708 void stop(void)
709 {
710   vel.linear.x = 0.0;
711   vel.linear.y = 0.0;
712   vel.linear.z = 0.0;
713   vel.angular.x = 0.0;
714   vel.angular.y = 0.0;
715   vel.angular.z = 0.0;
716   vel_pub.publish(vel);
717 }
718 int main(int argc, char** argv)
719 {
720   ros::init(argc,argv,"reach_point");
721   ReachPoint rp;
722   srand(time(NULL));
723   ros::spin();
724   return 0;
725 }
```

Glossary

A:

α	A unique point outside of the field. Point of entrance
APF	Artificial Potential Field

D:

d_f	distance between two consecutive furrow
$dist_{pl}$	Required ploughing distance
$dist_{ft}$	Required furrow transitioning distance
D_{pl}	Total ploughing distance
D_{ft}	Total furrow transitioning distance
d_{stage_1}	Travel distance in stage 1
d_{stage_2}	Travel distance in stage 2
d_{stage_3}	Travel distance in stage 3
d_{stage_4}	Travel distance in stage 4
$d_{\alpha l_1}$	Distance between α and the first track (spraying)
$d_{l_{j-1}l_j}$	Distance between two consecutive track locations (spraying)

E:

ϵ	Collision Threshold Distance

F:

FR	FRONT REGION
F_{sum}	Sum Force
F_{att}	Attraction Force
F_{rep}	Repulsion Force

G:

Γ	Detecting Range of a robot
G_i	Set of location ids allocated to robot i

H:

H	Headland Width
h_1, h_2	Headland width in each side

K:

K	Number of ploughing locations

L:

L	Length of the field
λ	Robot's Length
l_f	length of a single furrow
LR	LEFT REGION

M:

m	the last track number assigned to previous robots (spraying)

N:

n_{opt}	Optimum number of robots in the team (spraying)

P:

PC	Ploughing Cost

R:

$Region_n$	number of regions in the field (spraying)
r_i	the i'th robot
RR	RIGHT REGION

T:

τ_{init}	constant time to perceive task initiation event (spraying)
τ_{p_n}	Ploughing delay time
t_{delay}	Field accessing delay time
t_p	Single furrow ploughing time for a single furrow
T_p	Total ploughing time
$T_{ploughing}$	Team ploughing time
t_{s_i}	Standby period for robot i (spraying)
$t_{spraying_i}$	refers to spraying time for robot i (spraying)
t_{t_a}	Task allocation period for robot i (spraying)

U:

$U(q)$	Potential Function
$U_{att}(q)$	Attraction Potential Function
$U_{rep}(q)$	Repulsion Potential Function

V:

v	Robot's Velocity

W:

W	Width of the field
w_f	Width of a single furrow